INFORMATION AND CONTROL 49, 125–169 (1981)

Passes and Paths of Attribute Grammars

JOOST ENGELFRIET AND GILBERTO FILÈ

Department of Applied Mathematics, Twente University of Technology, Enschede, The Netherlands

Contents. Abstract, Introduction, 1. Terminology, 2. The relationship between passes and paths, 3. Exponential lower bounds, 4. Upper bounds, 5. Path languages, Conclusion, References.

An attribute grammar is pure (left-to-right) multi-pass if a bounded number of left-to-right passes over the derivation tree suffice to compute all its attributes. There is no requirement, as for the usual multi-pass attribute grammars, that all occurrences of the same attribute are computed in the same pass. It is shown that the problem of determining whether an arbitrary attribute grammar is pure multi-pass, is of inherently exponential time complexity. For fixed $k > 0$, it can be decided in polynomial time whether an attribute grammar is pure k-pass. The proofs are based on a characterization of pure multi-pass attribute grammars in terms of paths through their dependency graphs. A general result on dependency paths of attribute grammars relates them to (finite-copying) top-down tree transducers. The formal power of k-pass attribute grammars increases with increasing k. Formally, multi-pass attribute grammars are less powerful than arbitrary attribute grammars.

INTRODUCTION

A large part of this paper is a variation on a known theme. The theme was written by Jazayeri *et al.* (1975), determining the precise complexity of the circularity problem of attribute grammars. Recall that each derivation tree of an attribute grammar (AG) has an associated directed graph, called the dependency network of the tree, which indicates the dependencies between the attributes of all nonterminals in the tree, as given by the semantic rules. An attribute grammar is circular if there is a cycle in one of its dependency networks. The story of Jazayeri, Ogden and Rounds, which shows that the circularity problem is of intrinsically exponential time complexity, is based on the properties of paths in the dependency networks of an attribute grammar. In fact, they use "dependency paths" to simulate computations of exponential-time Turing machines, and they simulate dependency paths by the derivations of a context-free grammar (of exponential size) to detect cycles.

0019-9958/81/050125-45$02.00/0

In this paper we apply these path-techniques to (left-to-right) multi-pass attribute grammars. A pass is a left-to-right depth-first traversal of a derivation tree, during which attributes of the nodes of the tree are evaluated (Lewis *et al.*, 1974; Bochmann, 1976). Multi-pass attribute grammars, for which a bounded number of (left-to-right) passes over the tree suffices to compute all its attributes, were introduced by Bochmann (1976) as a formal model of multi-pass compilers. As observed by Alblas (1980), Bochmann, in his definition of multi-pass AG, made the more or less implicit assumption that different occurrences of the same attribute (of some nonterminal) should be evaluated in the same pass. Thus, in Alblas (1980) the multi-pass AG satisfying this restriction are called "simple," whereas the general (theoretically cleaner) multi-pass AG are called "pure." The pure multi-pass attribute grammars are the main subject of this paper.

Some more or less obvious differences between the pure and simple multi-pass AG are the following. First, every simple multi-pass AG is clearly also a pure multi-pass AG, but there are more pure than simple multi-pass AG, see Alblas (1980) for examples. On the other hand (as shown in this paper), every pure multi-pass AG can be transformed into a simple multi-pass AG which realizes the same translation (and uses the same semantic operations). Second, there is a trade-off in the time needed for attribute evaluation. For a simple multi-pass AG, the number of "pure passes" needed is in general less than the number of "simple passes" needed, but each pure pass may take more time, see again Alblas (1980) for examples.

The main aim of this paper is to show the difference between pure and simple in terms of the time needed to decide these properties. We prove that it is of inherently exponential time complexity to decide whether an arbitrary attribute grammar is pure multi-pass. Note that deciding whether an arbitrary attribute grammar is simple multi-pass takes polynomial time (Bochmann, 1980). On the other hand we prove that for fixed k the pure k-pass property is decidable in polynomial time.

To prove these results we first characterize (in Section 2) the pure multi-pass property in terms of dependency paths: an attribute grammar is pure multi-pass if and only if there is a bound on the number of "R-edges" in its dependency paths, where an R-edge is one which "runs from right to left," i.e., opposite to the direction of the pass. A similar characterization for the simple case is given in Alblas (1980) and, independently, in Räihä and Ukkonen (1980). Using this "path-characterization of passes" we present, as observed before, a variation of the paper of Jazayeri *et al.* (1975), proving the time complexity results (lower bounds in Section 3 and upper bounds in Section 4).

In the last section (Section 5) we take a more formal point of view on dependency paths in general. Representing each dependency edge as a symbol, every dependency path becomes a string, and the set of all depen-

dency paths of an attribute grammar can be viewed as a formal language. In general this language is not context-free. However, it turns out that every dependency path language can be generated as the output language of a top-down tree-to-string transducer, which is moreover "finite-copying" (see Engelfriet *et al.*, 1980). In other words (Engelfriet *et al.*, 1980) it is the output language of a deterministic finite-state tree-walking automaton (Aho and Ullman, 1971). The above time-complexity results can be viewed in the light of this relationship.

Another reason for the importance of dependency paths is the following. When investigating the formal power of different classes of attribute grammars it is necessary to abstract from the meaning of the basic operations on attribute values used in the semantic rules (just as is done in program scheme theory). In this way each string of the underlying context-free language is translated into a formal expression, i.e., a tree, and the set of these trees forms a tree language. We show that the dependency path language of an attribute grammar is closely related to the "output path language," i.e., the set of all paths through trees in the (output) tree language. In fact, the class of output path languages of arbitrary attribute grammars is precisely the class of output languages of the above-mentioned device: the finite-copying top-down tree-to-string transducer. Since the output path languages of multi-pass AG form a smaller class, it follows that (as expected) arbitrary attribute grammars are more powerful (in the "schematic" sense) than multi-pass attribute grammars. Moreover, the number of passes gives rise to a proper hierarchy (and the same is true for the number of "visits" of arbitrary AG).

Throughout the paper the reader is assumed to be familiar with attribute grammars (Knuth, 1968; Bochmann, 1976). In Section 1 some of the relevant terminology is collected. In Section 5 (only), the reader should be familiar with top-down tree transducers (Rounds, 1970a; Engelfriet, 1975).

1. Terminology

In this section we list some terminology and definitions concerning attribute grammars. We abbreviate "attribute grammar" by "AG." The empty string is denoted λ.

An *attribute grammar* G consists of (1)–(4) as follows.

(1) G has an *underlying context-free grammar* (N, T, P, Z), usually also denoted G, consisting of nonterminals, terminals, productions and initial nonterminal, respectively. A production $p \in P$ is denoted as p: $X_0 \to w_0 X_1 w_1 X_2 w_2 \cdots X_{n_p} w_{n_p}$, where $X_i \in N$ and $w_i \in T^*$, $n_p \geqslant 0$. A derivation tree of G usually has terminals (or λ) at its leaves; if not, we talk

about *partial* derivation trees; a derivation tree with root labeled Z (and terminal leaves) is said to be *complete*. We assume G to be a reduced context-free grammar in the sense that every production occurs in at least one complete derivation tree.

(2) Each nonterminal X of G has two disjoint finite sets, denoted $I(X)$ and $S(X)$, of *inherited and synthesized attributes*, respectively (shortly, *i*-attributes and *s*-attributes). The initial nonterminal Z has no *i*-attributes, and one of its *s*-attributes is *designated* to hold the meaning of any complete derivation tree. An attribute a of X is also denoted $a(X)$. Attributes of different nonterminals are different. Note that terminals do not have attributes.

(3) With each attribute a of G a set of possible *values* of a is associated, denoted $V(a)$.

(4) With each production $p \in P$ is associated a set of *semantic rules* which define all attributes in $S(X_0)$ and $I(X_j)$, $1 \leqslant j \leqslant n_p$. A semantic rule defining attribute $a_0(X_{i_0})$ has the form $a_0(X_{i_0}) := f(a_1(X_{i_1}),..., a_m(X_{i_m}))$ where $0 \leqslant i_j \leqslant n_p$ and f is a mapping from $V_1 \times \cdots \times V_m$ into V_0, with $V_j = V(a_j(X_{i_j}))$, $0 \leqslant j \leqslant m$. Usually f is specified by some expression which uses certain operations between the $V(a)$. We say that $a_0(X_{i_0})$ *depends* on $a_j(X_{i_j})$ in p.

We assume that G is in *normal form* in the sense that in any semantic rule of the above form each $a_j(X_{i_j})$ is either in $I(X_0)$ or in $S(X_k)$ for some k, $1 \leqslant k \leqslant n_p$, cf. Bochmann (1976).

The semantic rules are used to evaluate all attributes of the nonterminals in a complete derivation tree t. The value of the designated attribute of the root of t is the meaning of t (or better, the meaning of the string which is the yield of t). In this way G realizes a *translation* from strings (or trees) to values.

This ends the description of an attribute grammar.

In examples we allow attributes of different nonterminals to have the same name (although, by definition, they are different). To distinguish them we add the name of the nonterminal. Thus, if $a \in S(X_1)$ and $a \in S(X_2)$, then $a(X_1)$ and $a(X_2)$ are different attributes.

Most of our results will not be concerned with the actual meaning of the semantic rules but just with the dependencies they cause between the attributes. Thus we need the following more or less well-known concepts concerning dependencies (Knuth, 1968; Bochmann, 1976; Jazayeri *et al.*, 1975). Let G be an attribute grammar as above.

For production p, its *dependency graph*, denoted by $D(p)$, is the directed graph which has as nodes the attributes of all nonterminals X_j of p $(0 \leqslant j \leqslant n_p)$ and in which there is an edge from attribute a to attribute b if b depends on a in p. For a partial derivation tree t, its *dependency network*,

denoted by $D(t)$, is the directed graph obtained by putting together all dependency graphs of productions used in t (identifying the appropriate nodes). For an example, see Figs. 1 and 2a. An edge in a dependency graph or network is called a *dependency edge*. A (directed) path in a dependency network is called a *dependency path*. It is *acyclic* if its nodes are all different, otherwise it is cyclic or circular (in Jazayeri *et al.* (1975) "simple" is used instead of "acyclic"). A dependency network is acyclic if all its paths are acyclic. An attribute grammar is *noncircular* if all its dependency networks are acyclic, otherwise it is circular.

For a nonterminal X, an *is-graph* is a directed graph which has as nodes all attributes of X and in which each edge runs from an i-attribute to an s-attribute (called "i/o graph" in Kennedy and Warren (1976)). An *is*-graph will also be viewed as a finite set of pairs (a, b), where $a(b)$ is an i-(s-) attribute of X, cf. Jazayeri *et al.* (1975). A dependency path in some $D(t)$ is an *is-path* if it runs from an i-attribute of the root to an s-attribute of the root of t. For a derivation tree t, the *is-graph of* t, denoted by $is(t)$, is the *is*-graph of X (where X labels the root of t) in which an edge runs from i_0 to s_0 if and only if there is an *is*-path from i_0 to s_0 in $D(t)$. Thus the *is*-graph of t models all dependencies between the attributes of its root.

2. The Relationship between Passes and Paths

A *pass* is a left-to-right depth-first traversal of the derivation tree during which attributes of the nodes of the tree may be evaluated. More precisely it is defined by the following (nondeterministic) recursive procedure "pass-tree" which has a node of the tree as parameter, cf. Bochmann (1976).

```
proc pass-tree(x0); node x0;
        {let x0 have nonterminal sons x1, x2,..., xn}
    begin for i := 1 to n do
            evaluate some inherited attributes of xi;
            pass-tree(xi)
                od;
        evaluate some synthesized attributes of x0
    end.
```

With "some" is meant that nondeterministically certain (possibly no) attributes are chosen and evaluated by the appropriate semantic rules of the production $X_0 \to w_0 X_1 w_1 \cdots X_n w_n$ where X_i is the label of x_i (of course, under the requirement that all arguments of these semantic rules have been evaluated before).

A pass over the complete derivation tree consists of a possible execution

of the call pass-tree(root), where "root" is the root of the tree. In general several consecutive passes over the tree are needed to evaluate all its attributes. It should be clear that for every complete derivation tree t (with acyclic dependency network $D(t)$) there is an integer k such that all attributes of t can be evaluated in k passes, i.e., by the algorithm **for** $j := 1$ **to** k **do** pass-tree(root) **od**, where k depends in general on t. Note that this algorithm is still nondeterministic due to the presence of "some" in the body of pass-tree. Obviously (cf. the proof of Theorem 2.2) we could make it deterministic by replacing "some" by "as many ⋯ as possible" (i.e., all attributes which can be evaluated). The reason that we prefer such a general definition of pass (apart from the fact that theoreticians like nondeterminism) is that it allows for many different deterministic particularizations. As another example, replacing "some" by "all ⋯ in the set A_j," where A_j is a fixed set of attributes to be computed at the jth pass, gives the concept of pass used in the multi-pass attribute grammars of Bochmann (1976).

How many passes are needed to evaluate all attributes of a given complete derivation tree t? Intuitively, the number of passes is determined by the right-to-left dependencies in the dependency network of t.

2.1. DEFINITION. An edge in the dependency graph of a production $X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ is an *R-edge* if it runs from an s-attribute of X_k to an i-attribute of X_j with $1 \leqslant j \leqslant k \leqslant n_p$. Similarly, the R-edges of a dependency network of a derivation tree are those of the dependency graphs it is composed of. ∎

For a derivation tree t and a dependency path π in the dependency network $D(t)$ of t, we denote by $\#_R(\pi)$ the number of R-edges of π.

We now show that the number of passes needed to evaluate all attributes of t is one plus the maximal number of R-edges in the paths of $D(t)$.

2.2. THEOREM. *Let G be a noncircular attribute grammar. For every complete derivation tree t of G and every integer k, all attributes of t can be evaluated in k passes if and only if $k \geqslant \max\{\#_R(\pi) \mid \pi \text{ is a path in } D(t)\} + 1$.*

Proof. ($\Rightarrow$) Suppose that all attributes of t can be evaluated in k passes, i.e., there is a way of executing the nondeterministic algorithm **for** $j := 1$ **to** k **do** pass-tree(root) **od**, such that it computes all attributes of t. Denote, for every attribute a of t (i.e., node of $D(t)$), by pass(a) the number of the pass in which a is evaluated. Clearly, if there is an edge from b to a in $D(t)$, then pass(a) $\geqslant$ pass(b), because a depends on b; if the edge is moreover an R-edge, then pass(a) $>$ pass(b), because in any pass, a is considered earlier than b. Hence, if there is a path π from b to a in $D(t)$, then pass(a) $\geqslant$ pass(b) $+ \#_R(\pi)$, and so $k \geqslant$ pass(a) $\geqslant$ pass(b) $+ \#_R(\pi) \geqslant 1 + \#_R(\pi)$. Consequently, $k \geqslant \max\{\#_R(\pi) \mid \pi \text{ is a path in } D(t)\} + 1$.

($\Leftarrow$) It suffices to show this for $k = \max\{\#_R(\pi) | \pi \text{ is a path in } D(t)\} + 1$. For every attribute a of t, let $M(a) = \max\{\#_R(\pi) | \pi \text{ is a path in } D(t) \text{ leading to } a\}$. We will show that attribute a can be evaluated in pass $M(a) + 1$. In other words, all attributes of t can be evaluated in k passes if, when calling procedure pass-tree(root) for the jth time, "some" is replaced by "all $\cdots$ with $M(a) + 1 = j$." Since by the other direction of this proof, $\text{pass}(a) \geqslant M(a) + 1$, this amounts to the same thing as replacing "some" by "as many as possible," i.e., $M(a) + 1$ is the earliest pass in which a can be evaluated. The proof is by induction on $M(a)$. Assume that in the first j passes all attributes a of t with $M(a) < j$ have been computed. We have to show that the $(j + 1)$st call of pass-tree(root) can compute all attributes with $M(a) = j$. Again by induction (on the height of node x_0 in t) it can be shown that if the inherited attributes a of x_0 with $M(a) = j$ are computed, then the call pass-tree(x_0) can compute all other attributes a with $M(a) = j$ of the subtree rooted at x_0. The easy proof is based on the fact that if there is an R-edge from b to a (where a is an inherited attribute of x_i, $i \geqslant 1$, such that $M(a) = j$), then, by definition of M, $M(b) < M(a) = j$ and hence the value of b has been computed in a previous pass. The details are left to the reader (formally another induction is needed on i). ∎

By this theorem, the attributes of every noncircular AG can be evaluated by the algorithm **while** not all attributes are computed **do** pass-tree(root) **od**. For certain AG the number of repetitions of the while-statement will be bounded by a constant. This class of "pure" multi-pass AG was introduced by Alblas (1980).

2.3. DEFINITION (Alblas, 1980).

(i) Let k be an integer, $k \geqslant 1$. An AG G is *pure k-pass* if for every complete derivation tree t of G all attributes of t can be evaluated in k passes.

(ii) An AG is *pure multi-pass* if it is pure k-pass for some $k \geqslant 1$. ∎

A pure 1-pass AG is also called an L–AG.

From Theorem 2.2 we immediately obtain a characterization of the pure multi-pass AG: there should be a bound on the number of R-edges in the dependency paths of the grammar (and one plus this bound is the number of passes needed).

2.4. THEOREM. *Let G be an arbitrary AG and let $M(G)$ denote the (possibly infinite) number* $\max\{\#_R(\pi) \mid \pi$ *is a path in* $D(t)$ *for some complete derivation tree t of G}.

(i) *For $k \geqslant 1$, G is pure k-pass if and only if $M(G) + 1 \leqslant k$.*

(ii) *G is pure multi-pass if and only if $M(G)$ is finite.*

Proof. Immediately from Theorem 2.2. Note that, for a circular AG, $M(G)$ is infinite because every cycle in a dependency network contains at least one R-edge. ∎

In the following sections we will use the criteria of this theorem to show that the pure k-pass and multi-pass properties are decidable in polynomial and (inherently) exponential time, respectively.

As observed before, the class of pure multi-pass AG properly includes the class of (simple) multi-pass AG of Bochmann (1976). Actually there is a pure 2-pass AG which is not simple multi-pass (see Example 2.5); the pure 1-pass and simple 1-pass AG coïncide (the L-AG). The *simple multi-pass* AG are defined by requiring the existence of a partition $A_1, A_2, \ldots, A_k$ of the attributes such that for every derivation tree t all attributes of t in A_j can be evaluated in the jth pass. Thus, for simple multi-pass AG, different occurrences of one attribute in the same derivation tree (or in different derivation trees) have to be evaluated in the same pass. This difference between pure and simple was introduced and studied by Alblas (1980). It is proved in Alblas (1980), and independently in Räihä and Ukkonen (1980), that an AG is simple multi-pass if and only if there is a bound on the number of R-edges in paths in a certain dependency graph $D(G)$ obtained by merging all dependency graphs $D(p)$ of productions of G: the nodes of $D(G)$ are the attributes of all nonterminals of G and there is an (R-)edge from a to b if there is an (R-)edge from (an occurrence of) a to (an occurrence of) b in some $D(p)$. Let $M_s(G)$ be $\max\{\#_R(\pi) \mid \pi \text{ is a path in } D(G)\}$. Then (Alblas, 1980), G is simple multi-pass if and only if $M_s(G)$ is finite, and the number of (simple) passes needed is $M_s(G) + 1$; note that $M_s(G)$ is infinite iff $D(G)$ contains a cycle with at least one R-edge (Alblas, 1980; Räihä and Ukkonen, 1980). Thus both the simple and pure multi-pass property are characterized by finiteness of $\max\{\#_R(\pi)\}$, but in the pure case π is a path in any dependency network, whereas in the simple case π is a path in the graph obtained by merging all dependency networks, identifying all nodes which are different occurrences of the same attribute (and so $M(G) \leqslant M_s(G)$, i.e., the number of pure passes needed is in general less than the number of simple passes needed). It should be clear that the simple multi-pass and the simple k-pass properties are decidable in polynomial time ((Bochmann, 1976); see also (Alblas, 1980; Räihä and Ukkonen, 1980) for alternative algorithms).

2.5. EXAMPLE. Consider the attribute grammar G_0 with nonterminals $\{Z, C, A, B\}$, terminals $\{a, b\}$ and productions $Z \to C$, $C \to AB$, $B \to AB$, $A \to a$, and $B \to b$. The nonterminals have the following attributes: $S(Z) = \{s\}$, $I(A) = I(C) = \{i\}$, $S(A) = S(C) = \{s\}$, $I(B) = \{i_1, i_2\}$, and $S(B) = \{s_1, s_2\}$. The dependency graphs of the productions are given in

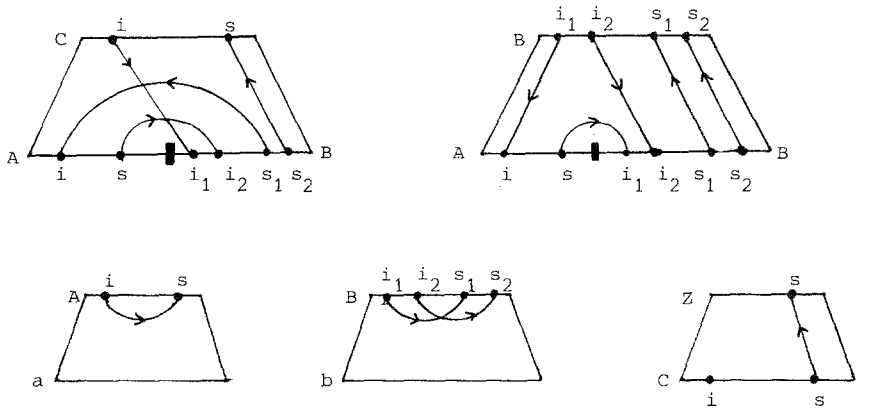

FIG. 1. Dependency graphs of a pure multi-pass AG G_0.

Fig. 1. Note that there is only one R-edge, in production $C \to AB$, which can occur only once in a derivation tree. Hence every dependency path contains at most one R-edge, and so G is pure 2-pass by Theorem 2.4. In fact, consider the dependency network of a derivation tree t, cf. Fig. 2a. In the first pass $i(C)$, all $i_1(B)$ and $s_1(B)$, and all $i(A)$ and $s(A)$ of nonterminals A occurring in the right subtree of t can be evaluated. In the second pass, the topmost $i(A)$ and $s(A)$ can be evaluated, and all $i_2(B)$, $s_2(B)$, and $s(C)$ and $s(Z)$. However, G is not simple multi-pass because it is impossible to evaluate all occurrences of $i(A)$ (and $s(A)$) in the same pass: in Fig. 2a, $i(A)$ of the topmost A depends on $i(A)$ of another A to its right. The dependency graph $D(G_0)$ of Alblas (1980) is shown in Fig. 2b. The only R-edge is contained in a cycle. ∎

Other examples can be found in Alblas (1980).

As an example of the use of Theorem 2.2 we show that the translational power of the pure multi-pass AG is the same as that of the simple multi-pass AG.

2.6. THEOREM. *For every pure k-pass AG G there is a simple k-pass AG G′ such that* (i) *G′ and G have the same underlying context-free grammar,* (ii) *G′ is equivalent to G in the sense that they assign the same meaning to every complete derivation tree, and* (iii) *G′ uses the same semantic rules as G, apart from the names of the attributes, and some additional semantic rules of the form a := c where c is a constant.*

Proof. G is changed into G' as follows. Each attribute a of a nonterminal X is changed into k attributes $\langle a, 1\rangle$, $\langle a, 2\rangle$,..., $\langle a, k\rangle$ for X of the same type (i– or s–). For each attribute a, let a_0 denote an arbitrary constant in $V(a)$.

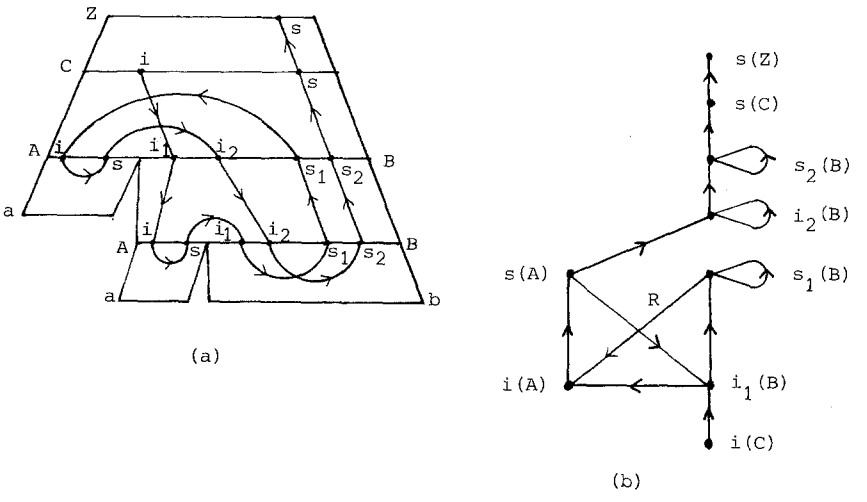

FIG. 2. (a) Dependency network of G_0. (b) Dependency graph $D(G_0)$.

A semantic rule $a := f(b_1,\ldots, b_m)$ such that no edge (b_n, a) is an R-edge, is changed into all semantic rules $\langle a,j\rangle := f(\langle b_1,j\rangle,\ldots, \langle b_m,j\rangle)$, $1 \leqslant j \leqslant k$, in particular if $m = 0$. A semantic rule in which at least one R-edge occurs is changed into the semantic rules $\langle a, 1\rangle := a_0$ and (for all $2 \leqslant j \leqslant k$) $\langle a,j\rangle := f(\langle b_1,j_1\rangle,\ldots, \langle b_m,j_m\rangle)$, where $j_n = j$ if (b_n, a) is not an R-edge, and $j_n = j - 1$ if (b_n, a) is an R-edge $(1 \leqslant n \leqslant m)$. Finally, if s is the designated attribute of G, then $\langle s, k\rangle$ is the one of G'.

It should be clear that every occurrence of an attribute $\langle a,j\rangle$ can be computed at the jth pass: since each R-edge adds one to the second component of the attributes, there can be at most $j - 1$ R-edges on a dependency path leading to $\langle a,j\rangle$, see the proof of Theorem 2.2. Hence G' is simple k-pass. Furthermore it should be clear that if an occurrence of an attribute a is computed in the jth pass in G, then the corresponding occurrences of $\langle a,j\rangle$, $\langle a,j+1\rangle,\ldots, \langle a,k\rangle$ will in G' all get the same value as attribute a (no "wrong" rule $\langle b, 1\rangle := b_0$ can ever be used to compute these attributes). Hence, in particular, every attribute $\langle a, k\rangle$ has the same value as attribute a. ∎

We finally mention that the simple/pure distinction can also be applied to visits. Simple multi-visit AG are considered in Engelfriet and Filè (1980) and shown to be the same as the linearly ordered AG (cf. Kastens, 1980); the simple multi-visit property turns out to be NP-complete (and so is the simple k-visit property for $k \geqslant 2$). Pure multi-visit AG are considered in Riis and Skyum (1980) and shown to coïncide with the noncircular AG; the pure k-visit property is decidable. Pure k-visit AG will be considered in Section 5.

3. Exponential Lower Bounds

In this section we prove that the same exponential lower bound which holds for the circularity problem (Jazayeri *et al.*, 1975) is also valid for the pure multi-pass problem. Thus, there is a constant c such that any deterministic algorithm which decides whether an attribute grammar is pure multi-pass, runs for more than $2^{cn/\log n}$ steps on infinitely many attribute grammars G (where n is the size of G). Instead of proving this directly we will show the lower bound for a particular problem concerning is-graphs of L-AG. This problem can then be reduced to both the circularity problem and the pure multi-pass problem (and some others). This means that exponential lower bounds are caused by the difficulty of computing *is*-graphs (even for L-AG). The proof of the theorem follows the ideas of Jazayeri *et al.* (1975), simplified by the use of the linear space alternating Turing machine (ATM, see (Chandra *et al.*, 1980)) rather than the linear space auxiliary pushdown automaton (Cook, 1971) as a characterization of the class $\bigcup_d \mathrm{DTIME}(2^{dn})$ of exponential-time languages. A proof for the circularity problem which reduces every exponential-time Turing machine directly to the circularity problem is given in Jones (1980); a slight modification of this proof could also be used for the next theorem.

3.1. Theorem. *Let K be an exponential-time language, i.e., $K \in \mathrm{DTIME}(2^{dn})$ for some d. For every word w of length n there is an attribute grammar G_w such that*

(1) *G_w is an L-AG (i.e., it is 1-pass);*

(2) *G_w can be constructed in deterministic time $O(n \log n)$, where the constant depends on K only;*

(3) *a designated nonterminal B of G_w has i-attributes $i_1, i_2, \ldots, i_n$ and s-attributes $s_1, s_2, \ldots, s_n$;*

(4) *$w \in K$ if and only if there is a derivation tree t of G_w with root labeled B such that its is-graph $is(t)$ has edges running from i_k to s_k for all k, $1 \leqslant k \leqslant n$, see Fig. 3;*

(5) *for every derivation tree t of G_w with root B, if e is an edge of $is(t)$, then e runs from i_k to s_k for some k, $1 \leqslant k \leqslant n$ (i.e., $is(t)$ is a subgraph of the graph of Fig. 3).*

Proof. Consider a one-tape linear space alternating Turing machine M recognizing K, which exists by Corollary 3.5 of Chandra *et al.* (1980). We assume that the length of the tape is exactly n, where n is the size of the input string. We consider a slight variation of the ATM, which can easily be shown equivalent to the original one. Each state q_1 of M may be either

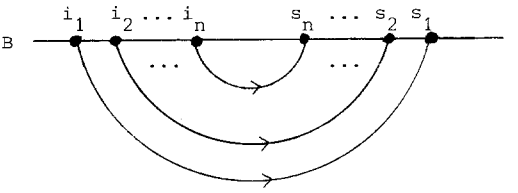

FIG. 3. Maximal *is*-graph of derivation tree with root B.

(i) an existential state, with one instruction of the form $q_1 \to q_2$ *or* q_3, where q_2 and q_3 are states of M (nondeterministic choice of new state q_2 or q_3, no changes on the tape), or

(ii) a universal state, with one instruction of the form $q_1 \to q_2$ *and* q_3, where q_2 and q_3 are states of M (parallel execution both in state q_2 and in state q_3, no changes on the tape), or

(iii) a deterministic state, with for each tape symbol x one instruction of the form $(q_1, x) \to (q_2, y, d)$ where q_2 is a state, y a tape symbol and $d \in \{\text{left, right}\}$ (when M reads x in state q_1, it prints y and moves in direction d in state q_2), or

(iv) an accepting state, with no instruction (M halts and accepts). M rejects by staying in an infinite computation.

The attribute grammar G_w will be constructed such that a derivation tree t of G_w together with its dependency network $D(t)$ represents an accepting computation tree of M. An *accepting computation tree* of M (Ladner *et al.*, Ruzzo, 1979) is a tree of accepting parallel computations. Each node of the tree is labeled by a configuration $\langle q_1, j, a_1 a_2 \cdots a_n \rangle$, where q_1 is a state, j is the position of the read/write head (reading a_j), and $a_1 a_2 \cdots a_n$ is the tape contents (a_k is a tape symbol for all k, $1 \leqslant k \leqslant n$). In cases (i) and (iii) above, the node has one successor, in case (ii) two successors, and in case (iv) it is a leaf. We say that a configuration is *successful* if it is the root of an accepting computation tree. The string w is accepted by M if the initial configuration $\langle q_0, 1, w \rangle$ is successful, where q_0 is the initial state of M.

G_w has all states of M as nonterminals, and two additional nonterminals B and Z. If Σ is the tape alphabet of M, then the terminal alphabet of G_w is Σ. Each state has i-attributes $L(j, a)$ and s-attributes $R(j, a)$, for all $-n+1 \leqslant j \leqslant n-1$ and $a \in \Sigma$; B has i-attributes $i_1, i_2, \ldots, i_n$ and s-attributes $s_1, s_2, \ldots, s_n$, and the initial nonterminal Z has no attributes. The productions and semantic rules of G_w are constructed corresponding to the instructions of M in such a way that an accepting computation tree whose root is labeled by configuration $\langle q, j, a_1 \cdots a_n \rangle$ is represented by a derivation tree t of G_w with root labeled q and with the property that $is(t)$ contains edges from $L(k-j, a_k)$ to $R(k-j, a_k)$ for all k, $1 \leqslant k \leqslant n$. Thus the state of the

configuration labels the root, a dependency path between the attributes $L(0, a)$ and $R(0, a)$ of the root is used to indicate the tape symbol a under the read/write head, whereas the tape content to the left (right) of the head is modeled by dependency paths from $L(u, a)$ to $R(u, a)$ with $u < 0$ ($u > 0$, respectively).

We now describe the productions and semantic rules of G_w. The attributes of q_i will be indicated by L_i and R_i.

(i) Corresponding to instruction $q_1 \to q_2$ *or* q_3 there are two productions $q_1 \to q_2$ and $q_1 \to q_3$ in G_w. The attribute values are just passed. For $q_1 \to q_2$ the semantic rules are $L_2(j, a) := L_1(j, a)$ and $R_1(j, a) := R_2(j, a)$ for all $-n + 1 \leqslant j \leqslant n - 1$ and $a \in \Sigma$; and similarly for $q_1 \to q_3$.

(ii) Corresponding to instruction $q_1 \to q_2$ *and* q_3 there is one production $q_1 \to q_2 q_3$ in G_w. The attribute values are just passed in a left-to-right fashion: $L_2(j, a) := L_1(j, a)$, $L_3(j, a) := R_2(j, a)$ and $R_1(j, a) := R_3(j, a)$, for all j and a.

(iv) Corresponding to an accepting state q_1 there is one production $q_1 \to \lambda$ in G_w (where λ is the empty string). The attribute values are just passed: $R_1(j, a) := L_1(j, a)$ for all j and a.

For the dependency graphs of (i), (ii) and (iv), see the symbolic pictures in Fig. 4a.

(iii) This is the most involved case. Corresponding to each instruction $(q_1, x) \to (q_2, y, d)$ there is one production $q_1 \to x q_2$ in G_w with the following semantic rules (where c is some constant attribute value).

In case $d =$ right, the L-attributes are shifted -1 and the R-attributes $+1$; $L_1(0, x)$ is passed to $L_2(-1, y)$ and $R_2(-1, y)$ is passed back to $R_1(0, x)$. Formally the rules are

$$\begin{cases} L_2(j, a) := L_1(j + 1, a) & \text{for } a \in \Sigma, -n + 1 \leqslant j \leqslant n - 1, \\ & j \neq -1, j \neq n - 1, \\ L_2(n - 1, a) := c & \text{for } a \in \Sigma, \\ L_2(-1, y) := L_1(0, x) & \\ L_2(-1, a) := c & \text{for } a \neq y, a \in \Sigma, \end{cases}$$

$$\begin{cases} R_1(j, a) := R_2(j - 1, a) & \text{for } a \in \Sigma, -n + 1 \leqslant j \leqslant n - 1, \\ & j \neq 0, j \neq -n + 1, \\ R_1(-n + 1, a) := c & \text{for } a \in \Sigma, \\ R_1(0, x) := R_2(-1, y) & \\ R_1(0, a) := c & \text{for } a \neq y, a \in \Sigma. \end{cases}$$

For an example of a dependency graph, see Fig. 4b.

In case $d =$ left, the L-attributes are shifted $+1$ and the R-attributes -1;

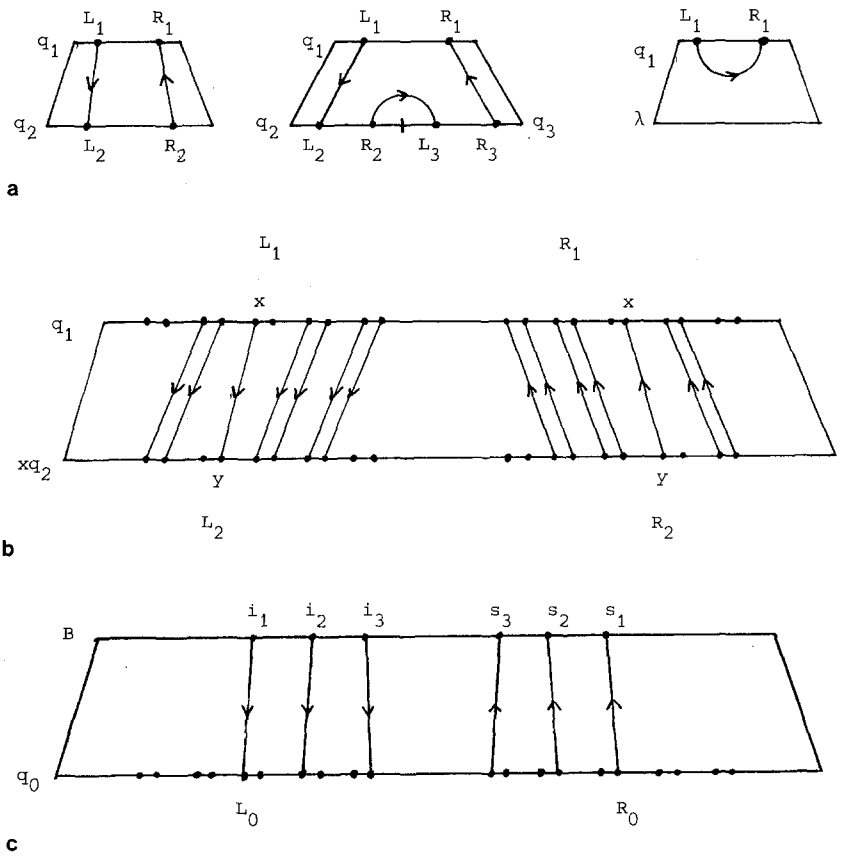

FIG. 4. (a) Dependency graphs for (i), (ii) and (iv), respectively; L stands for any $L(j, a)$, R for any $R(j, a)$. (b) Dependency graph for (iii), $\Sigma = \{a, b\}$, $n = 3$, $d =$ right, $x = a$, $y = b$; the attributes are displayed from left to right: $L(-2, a)$, $L(-2, b)$, $L(-1, a)$, $L(-1, b)$, $L(0, a)$, $L(0, b)$, $L(1, a)$,..., $L(2, b)$ and for R just vice versa $R(2, b)$, $R(2, a)$,..., $R(-2, b)$, $R(-2, a)$. (c) Dependency graph for $B \to q_0$; $\Sigma = \{a, b\}$, $n = 3$, $w = aab$. The attributes of q_0 are displayed as in Fig. 4(b).

$L_1(0, x)$ is passed to $L_2(1, y)$, and $R_2(1, y)$ to $R_1(0, x)$. The details of this case are left to the reader.

Finally, G_w has productions $B \to q_0$ and $Z \to B$, where B is the designated nonterminal of G_w, q_0 the initial state of M, and Z the initial nonterminal of G_w. If $w = w_1 w_2 \cdots w_n$ (where w_k is a tape symbol), then the semantic rules for $B \to q_0$ are

$$\begin{cases} L_0(k-1, w_k) := i_k & \text{for } 1 \leqslant k \leqslant n, \\ L_0(j, a) := c & \text{for all other values of } j \text{ and } a, \\ s_k := R_0(k-1, w_k) & \text{for } 1 \leqslant k \leqslant n \end{cases}$$

(see Fig. 4c for a dependency graph). The semantic rules for the production $Z \to B$ are $i_k := c$ for all $1 \leqslant k \leqslant n$ (this production is just there for completeness sake).

This ends the construction of G_w. In Fig. 5 an example is given of a dependency network of a derivation tree corresponding to an accepting computation tree with initial configuration $\langle q_0, 1, aab \rangle$. Note that this derivation tree also corresponds to the accepting computation tree with initial configuration $\langle q_0, 1, aaa \rangle$, because the third symbol is never read by M. The reader may compare Fig. 5 with Fig. 10 of Jazayeri *et al.* (1975).

We now prove that G_w satisfies all requirements. Clearly G_w is an L-AG. Also, the size of G_w is $O(n \log n)$, where the factor $\log n$ is needed to name each of the $O(n)$ attributes; see Jazayeri *et al.* (1975) for more details. Clearly G_w can be constructed in time $O(n \log n)$ by a deterministic log-space Turing machine. It remains to show points (4) and (5) in the statement of the theorem. It is easy to prove (by induction on the height of t) that, for a derivation tree t of G_w (with a state at its root), every edge of $is(t)$ runs from $L(j, a)$ to $R(j, a)$ for some $-n + 1 \leqslant j \leqslant n - 1$, $a \in \Sigma$. Hence, by the semantic rules of $B \to q_0$ (Fig. 4c), for a derivation tree t with root B, every edge of $is(t)$ runs from i_k to s_k for some k $(1 \leqslant k \leqslant n)$, which proves point (5). Finally, using this property of the $is(t)$ graphs, it is straighforward to prove (by induction on the height of the accepting computation tree or the derivation tree) that $\langle q, j, a_1 \cdots a_n \rangle$ is a succesful configuration of M if and only if there is a derivation tree t of G_w with root q such that there is an edge

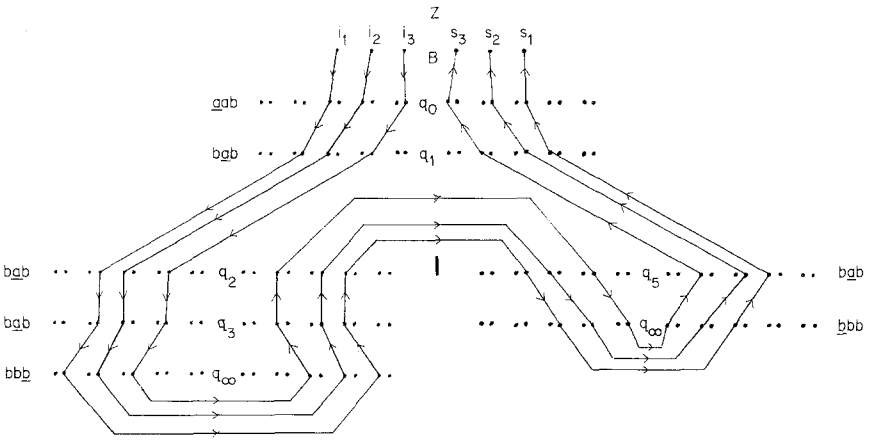

FIG. 5. Dependency network for an accepting computation tree; only the dependency paths from i_k to s_k are drawn $(1 \leqslant k \leqslant 3)$; $\Sigma = \{a, b\}$; initial configuration is $\langle q_0, 1, aab \rangle$; instructions of M are $(q_0, a) \to (q_1$, right, $b)$, $q_1 \to q_2$ *and* q_5, $q_2 \to q_3$ *or* q_4, $(q_3, a) \to (q_\infty$, right, $b)$, $(q_5, a) \to (q_\infty$, left, $b)$; q_∞ is an accepting state. The configuration at each node is indicated by the tape contents, with the scanned symbol underlined.

from $L(k-j, a_k)$ to $R(k-j, a_k)$ in $is(t)$, for every k $(1 \leqslant k \leqslant n)$. Hence (again by the semantic rules of $B \to q_0$), $w_1 \cdots w_n$ is accepted iff $\langle q_0, 1, w_1 \cdots w_n \rangle$ is succesful iff there is a derivation tree t with root B such that there is an edge from i_k to s_k for all k, $1 \leqslant k \leqslant n$. This proves point (4) and the theorem.

We note that the underlying context-free grammar of G_w is in general not reduced; this can be taken care of by adding productions $Z \to \#A$ and $A \to \#$ for every nonterminal A (where $\#$ is a new terminal) with empty dependency graphs. We finally note that if we require M to read all of its input, then there is a one-to-one correspondence between accepting computation trees and derivation trees with maximal is-graphs. ∎

Theorem 3.1 can be used to obtain exponential lower bounds for a number of problems, in particular the pure multi-pass problem. Due to point (2) in Theorem 3.1 this lower bound can be taken as $2^{cn/\log n}$ for some c (cf. Jazayeri *et al.* (1975) or Jones (1980) for the reasoning involved). For simplicity we use the following ad hoc terminology.

3.2. DEFINITION. A problem $P(x)$ is *exponential-time hard* to decide, if there is a constant $c > 0$ such that any deterministic Turing machine which decides whether an arbitrary x has property P must run for more than $2^{cn/\log n}$ steps for infinitely many x, where n is the size of x. ∎

In other words, problem P is exponential-time hard iff there is a constant $c > 0$ such that P is not in DTIME$(2^{cn/\log n})$. Note that this implies, e.g., that P is not in DTIME$(2^{dn^{1-\epsilon}})$ for any $d, \varepsilon > 0$.

What we have proved in fact in Theorem 3.1 is that it is exponential-time hard to decide whether a given L-AG has a given *is*-graph (so even L-AG are hard to analyze!).

3.3. COROLLARY. *It is exponential-time hard to decide for an arbitrary L-AG G, an arbitrary nonterminal B of G and an arbitrary is-graph g of B, whether there is a derivation tree t of G with root labeled B such that $is(t) = g$.*

Proof. By Theorem 3.1 every exponential-time language K can be reduced to this problem (in time $O(n \log n)$): from w construct G_w, B and the *is*-graph of B given in Fig. 3. See Jazayeri *et al.* (1975) for the fact that this implies exponential-time hardness. ∎

3.4. THEOREM. *It is exponential-time hard to decide each of the following problems:*

(1) *for an arbitrary AG, whether it is circular (Jazayeri et al., 1975);*
(2) *for an arbitrary pure multi-pass AG, an arbitrary nonterminal A*

of G, and an arbitrary i-attribute i_0 and s-attribute s_0 of A, whether there is a derivation tree t of G with root labeled A such that is(t) contains an edge from i_0 to s_0;

(3) *for an arbitrary noncircular AG, whether it is pure multi-pass.*

Proof. (1) Using the construction of Theorem 3.1, change G_w into $G_w^{(1)}$ by adding to G_w a production $A \to B$, where A is the (new) initial nonterminal of $G_w^{(1)}$ (without attributes). The semantic rules of $A \to B$ are $i_k := s_{k-1}$ for $2 \leqslant k \leqslant n$, and $i_1 := s_n$, see Fig. 6a. Since G_w is an L-AG, it should be clear that $G_w^{(1)}$ is circular only if the new production $A \to B$ gives rise to a cycle and, due to property (5) of Theorem 3.1, this is possible only if there are edges from i_k to s_k for all k. Hence, by property (4) of Theorem 3.1, $w \in K$ if and only if $G_w^{(1)}$ is circular, and we have reduced K to the circularity problem. The additional time needed to add $A \to B$ is again $O(n \log n)$.

(2) We use an obvious variant of the trick in (1). This time G_w is changed into $G_w^{(2)}$ by adding a nonterminal A with attributes i_0 and s_0, and a production $A \to B$ with semantic rules $i_k := s_{k-1}$ for $2 \leqslant k \leqslant n$ (as in (1)), $i_1 := i_0$, and $s_0 := s_n$ (thus the rule $i_1 := s_n$ of (1) is cut into two parts), see Fig. 6b. For completeness sake a production $Z \to A$ is added with semantic rule $i_0 := c$. Clearly, again by property (5) of Theorem 3.1, there is a path from i_0 to s_0 in some dependency network if and only if there are paths from i_k to s_k for all k. Hence we have reduced K to problem (2), still in time $O(n \log n)$.

It is easy to see that $G_w^{(2)}$ is pure multi-pass. Formally we can use the criterion of Theorem 2.4. Since G_w is L-AG (i.e., 1-pass), the only R-edges

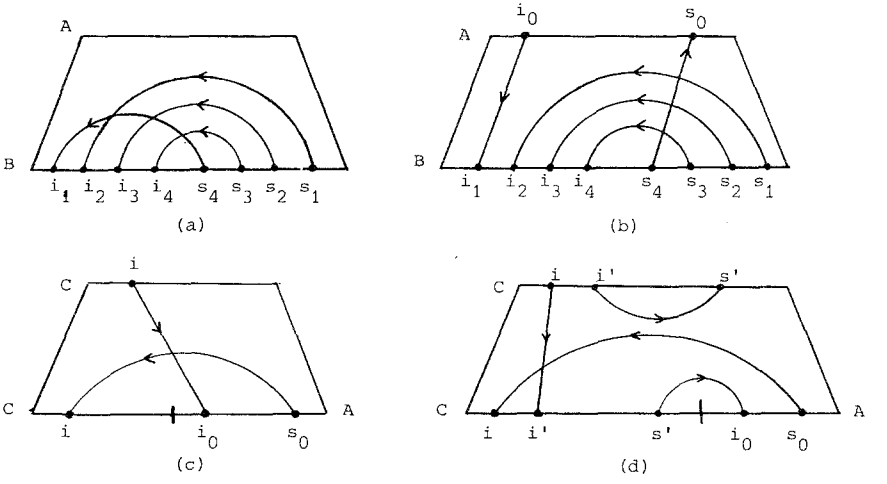

FIG. 6. Dependency graphs used in reduction proofs; $n = 4$ in (a) and (b).

of $G_w^{(2)}$ are the $n-1$ R-edges of production $A \to B$. Since, by property (5) of Theorem 3.1, $G_w^{(2)}$ is noncircular, $\max\{\#_R(\pi) \mid \pi$ is a path in some $D(t)\} \leqslant n-1$, and so $G_w^{(2)}$ is pure n-pass.

(3) We now change the pure multi-pass AG $G_w^{(2)}$ obtained in the proof of (2) into a noncircular AG $G_w^{(3)}$ by adding two nonterminals Z_3 and C, and three productions $Z_3 \to C$, $C \to CA$ and $C \to \lambda$, where A is of course the nonterminal of $G_w^{(2)}$ mentioned in (2). The new initial nonterminal Z_3 has no attributes, C has one i-attribute i. Production $Z_3 \to C$ has semantic rule $i := c$ (where c is a constant), production $C \to \lambda$ has no semantic rules, and production $C_1 \to C_2 A$ has the semantic rules $i(C_2) := s_0(A)$ and $i_0(A) := i(C_1)$, see Fig. 6c.

Clearly, since $G_w^{(2)}$ is noncircular, so is $G_w^{(3)}$. We now show that $G_w^{(3)}$ is not pure multi-pass if and only if $is(t)$ contains an edge from i_0 to s_0 for some derivation tree t with root labeled A, thereby reducing problem (2) to problem (3).

Assume first that there is such a derivation tree t with root A. Since the production graph of $C \to CA$ contains an R-edge from $s_0(A)$ to $i(C_2)$, we can build a complete derivation tree t_1 of $G_w^{(3)}$ with n R-edges (for any n) using $Z_3 \to C$ once, $C \to CA$ n times, $C \to \lambda$ once, and using t for all occurrences of A. Hence $\max\{\#_R(\pi) \mid \pi$ is a path in $D(t_1)$ for some complete derivation tree t_1 of $G_w^{(3)}\}$ is infinite, and so, by Theorem 2.4, $G_w^{(3)}$ is not pure multi-pass.

Assume now that there is no path from i_0 to s_0 in any dependency network. Consider an arbitrary complete derivation tree t of $G_w^{(3)}$ and an arbitrary path π in $D(t)$. If π runs completely inside a subtree with root A, then $\#_R(\pi) \leqslant n-1$, because $G_w^{(2)}$ is pure n-pass. Otherwise π can only run from one subtree with root A, via the R-edge from s_0 to i and the edge from i to i_0, into another subtree with root A, one level lower; hence $\#_R(\pi) \leqslant (n-1) + 1 + (n-1) = 2n-1$. Consequently, again by Theorem 2.4, $G_w^{(3)}$ is pure $2n$-pass.

This proves the reduction of (2) to (3), and the theorem. ∎

If also right-to-left passes are allowed, an attribute grammar whose attributes can be evaluated in a bounded number of passes, is called a pure alternating multi-pass AG (Alblas, 1980); Jazayeri and Walter (1975), where the simple alternating multi-pass AG are introduced). It is easy to modify the proof of Theorem 3.4(3) in order to show that also the pure alternating multi-pass property is exponential-time hard to decide: for production $C \to CA$ the dependency graph of Fig. 6d is taken and for production $C \to \lambda$ the semantic rule $s'(C) := i'(C)$ is used; if there is no connection between i_0 and s_0, the grammar is still pure $2n$-pass; if there is a connection between i_0 and s_0, then it is not difficult to see that the grammar is not pure alternating multi-pass (there are dependency paths with an unbounded number of alternations between R-edges and "L-edges"). A characterization of pure alter-

nating multi-pass AG in terms of dependency paths could be given analogous to Theorem 2.4. (cf. Alblas (1980) for the simple case).

Problem (2) of Theorem 3.4 shows that it is exponential-time hard for a given attribute grammar G and a given attribute a of G to decide whether a is "useless" [20] in the sense that it is never used to compute the value of the designated s-attribute of the root (which constitutes the "meaning" of the tree).

Problem (2) can be decided in polynomial time for L-AG; in fact, the wrong (polynomial-time) algorithm of Knuth (1968) for computing *is*-graphs is correct for L-AG, in the sense that it computes the overlapping of all possible *is*-graphs.

All three problems of Theorem 3.4 can actually be decided (for arbitrary AG) in exponential time, i.e., time 2^{dn} for some $d > 0$. For circularity, this is proved in Jazayeri *et al.* (1975). For the second problem, this follows immediately from the same proof in Jazayeri *et al.* (1975): all information concerning the existence of dependency paths from i- to s-attributes is incorporated in the decision method described in that proof. In the next section we will use the "path-technique" of that proof to show that also the pure multi-pass property can be decided in exponential time.

4. Upper Bounds

In Jazayeri *et al.* (1975) noncircularity is shown to be decidable in exponential time, i.e., in time 2^{dn} for some $d > 0$. In that paper, for each attribute grammar G a context-free grammar H is constructed (in exponential time) which "generates all acyclic is-paths" of G in the following sense. For every nonterminal X_0 of G and attributes $i_0(X_0)$ and $s_0(X_0)$, H has a nonterminal $(X_0, \{(i_0, s_0)\})$ such that $(X_0, \{(i_0, s_0)\}) \Rightarrow^* \lambda$ if and only if there is an acyclic *is*-path from i_0 to s_0 for some derivation tree of G with root X_0. It then suffices to reduce the grammar H (in polynomial time) in order to see whether cycles exist.

In this section we use the same technique to show the decidability of the pure multi-pass property (in exponential time) and of the pure k-pass property (in polynomial time). Instead of generating λ for each *is*-path π, we will generate a string with $\#_R(\pi)$ symbols.

4.1. Definition. The *R-language* of an attribute grammar G, denoted by $R(G)$, is the language over the alphabet $\{R\}$ defined by $R(G) = \{R^m \mid m = \#_R(\pi)$ for some path π in a dependency network of $G\}$. ∎

By Theorem 2.4 (where the maximum length of a string in $R(G)$ is denoted by $M(G)$) an attribute grammer is pure multi-pass if and only if its

R-language is finite. Therefore, to decide the pure multi-pass property for an attribute grammar G, we will, roughly speaking, construct (in exponential time) a context-free grammar H which generates the R-language $R(G)$ and then test finiteness of $L(H)$ in the usual way (in polynomial time). For the pure k-pass property it turns out that it suffices to consider a subgrammar of H which can be constructed in polynomial time.

Let us first consider how to generate all $\#_R(\pi)$ for acyclic *is*-paths π. Note that for the pure multi-pass property we would be able to assume all paths to be acyclic by first testing the grammar for noncircularity (by the result of Jazayeri *et al.* (1975)) and rejecting it if circular; however, for the k-pass property this takes too much time and therefore we will not assume it in general. To generate $\#_R(\pi)$ for an acyclic *is*-path π in some $D(t)$ we will simulate the derivation corresponding to t and keep information concerning π in the nonterminals, cf. Jazayeri *et al.* (1975). For each subtree t' of t with root X we keep track of how π "visits" t', i.e., of the *is*-graph D of X, where $D = \{(i_1, s_1) \mid \pi$ has a subpath which is an *is*-path from i_1 to s_1 in $D(t')\}$. In this way we can determine locally the number of R-edges involved in π.

4.2. LEMMA. *For every attribute grammar G a context-free grammar H can be constructed in exponential time (i.e., in time 2^{dn} for some $d > 0$, where n is the size of G) such that*

(i) *H has nonterminals of the form (X, D) where X is a nonterminal of G and D is an is-graph of X; the terminal alphabet of H is $\{R\}$;*

(ii) *for every nonterminal X_0 of G and attributes $i_0(X_0)$ and $s_0(X_0)$, $(X_0, \{(i_0, s_0)\}) \Rightarrow^* R^m$ if and only if $\#_R(\pi) = m$ for some acyclic is-path π from i_0 to s_0 in the dependency network of a derivation tree with root X_0.*

Proof. To explain the construction of H in detail we need the technical concept of an *augmented dependency graph or network*, defined as follows. For a production $p: X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$, the augmented dependency graph of p is obtained from $D(p)$ by adding edges (i, s) for all $i \in I(X_j)$ and $s \in S(X_j)$, $1 \leqslant j \leqslant n_p$. For a partial derivation tree t, the augmented dependency network of t is obtained from $D(t)$ by adding all edges (i, s) where $i \in I(X)$ and $s \in S(X)$ for every leaf of t labeled X. The augmented edges are not R-edges. The notion of *is*-path can easily be extended to augmented dependency networks. Note that what we did is adding maximal *is*-graphs to all nonterminal leaves of the tree.

The productions of the context-free grammar H will be constructed in such a way that the following property (*) can be shown for its derivations:

(*) $(X_0, \{(i_0, s_0)\}) \Rightarrow^* R^{m_0}(X_1, D_1) R^{m_1}(X_2, D_2) R^{m_2} \cdots (X_r, D_r) R^{m_r}$ in H with $m_0 + m_1 + \cdots + m_r = m$

if and only if

there is an acyclic *is*-path π in the augmented dependency network of the partial derivation tree corresponding to a derivation $X_0 \Rightarrow^* v_0 X_1 v_1 X_2 v_2 \cdots X_r v_r$ in G (where the v_i are terminal strings) such that

(i) $\#_R(\pi) = m$

(ii) π runs from $i_0(X_0)$ to $s_0(X_0)$

(iii) D_j is the set of edges (i, s) of π with $i \in I(X_j)$ and $s \in S(X_j)$, for all $1 \leqslant j \leqslant r$.

Thus, in derivations of H part of a path π_0 from $i_0(X_0)$ to $s_0(X_0)$ has been constructed (i.e., the m R-edges of that part have been generated), whereas the rest of π_0 is "predicted" in the *is*-graphs D_j.

The productions of H are constructed as follows. Choose a production $p: X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ of G. Choose a set $\{\pi_1, \pi_2, \ldots, \pi_v\}$ of paths ($v \geqslant 0$) in the augmented dependency graph of p such that (1) each π_k ($1 \leqslant k \leqslant v$) runs from an i-attribute of X_0 to an s-attribute of X_0, and (2) each node of the augmented dependency graph occurs at most once in $\pi_1, \pi_2, \ldots, \pi_v$ (i.e., they are disjoint acyclic paths). Now construct production $(X_0, D_0) \to R^m(X_1, D_1)(X_2, D_2) \cdots (X_{n_p}, D_{n_p})$ of H such that

(i) m is the total number of R-edges occurring in $\pi_1, \ldots, \pi_v$, i.e., $M = \sum_k \#_R(\pi_k)$;

(ii) $D_0 = \{(i_k, s_k) \mid \pi_k$ runs from $i_k(X_0)$ to $s_k(X_0)$, $1 \leqslant k \leqslant v\}$, and

(iii) for $1 \leqslant j \leqslant n_p$, D_j is the set of edges (i, s) occurring in $\pi_1, \ldots, \pi_v$ with $i \in I(X_j)$ and $s \in S(X_j)$.

For each such choice of production p and set $\{\pi_1, \ldots, \pi_v\}$ a production of H should be constructed.

It is straighforward to show that property (*) holds, using induction on the length of the derivation (considering the last production applied). For $r = 0$, property (*) of course implies point (ii) in the statement of the lemma. It remains to show that H can be constructed in exponential time (see Jazayeri *et al.* (1975) for a similar reasoning). Consider a path π_k from the set $\{\pi_1, \ldots, \pi_v\}$ of paths in the augmented dependency graph of p, as above. Clearly π_k consists of an alternating sequence of dependency edges and augmented edges, and therefore, π_k consists of a sequence $a_1, a_2, \ldots, a_{2u-1}, a_{2u}$ of attributes ($u \geqslant 1$) such that (a_{2j-1}, a_{2j}) is a dependency edge and (a_{2j}, a_{2j+1}) an augmented edge; thus $a_1, a_2, \ldots, a_{2u}$ all occur in the semantic rules of p. Hence, since all nodes of $\pi_1, \ldots, \pi_v$ occur in the semantic rules of p and they are disjoint acyclic paths, each set $\{\pi_1, \ldots, \pi_v\}$ can be written down in the same space as the semantic rules of p. Clearly, the corresponding production of H needs at most three times that amount of space plus space occupied by production p. It follows from this that the size of H is at most

2^{dn} for some $d > 0$, where n is the size of G. H can be constructed in time proportional to its size. ∎

4.3. EXAMPLE. To illustrate the construction in Lemma 4.2, consider the AG G_0 of Example 2.5. Let D_C be the *is*-graph $\{(i, s)\}$ of C, D_A the *is*-graph $\{(i, s)\}$ of A, and D_B the *is*-graph $\{(i_1, s_1), (i_2, s_2)\}$ of B. Then the following are productions of the context-free grammar H, obtained from the productions $C \to AB$, $B \to AB$, $A \to a$, and $B \to b$ of G_0, respectively:

(a) $(C, D_C) \to R(A, D_A)(B, D_B)$,

(b) $(B, D_B) \to (A, D_A)(B, D_B)$,

(c) $(A, D_A) \to \lambda$,

(d) $(B, D_B) \to \lambda$.

There are many more productions in H, but these are the ones which occur in derivations from (C, D_C), i.e., needed to generate all *is*-paths from $i(C)$ to $s(C)$. In Fig. 7a, b one can clearly see the paths in the augmented dependency graphs, used to obtain the above productions (a) and (b); the dotted lines are the augmented edges belonging to D_A and D_B (which are, by incidence, all augmented edges); see Fig. 1 for the productions (c) and (*d*). Clearly (C, D_C) generates the language $\{R\}$ which is in fact the R-language of G_0. ∎

The next theorem states the exponential upper bound for the pure multi-pass property.

4.4. THEOREM. *It is decidable whether an arbitrary attribute grammar G is pure multi-pas, in time 2^{dn} for some $d > 0$, where n is the size of G.*

Proof. To decide the pure multi-pass property we first reduce the problem of considering all dependency paths to the problem of considering

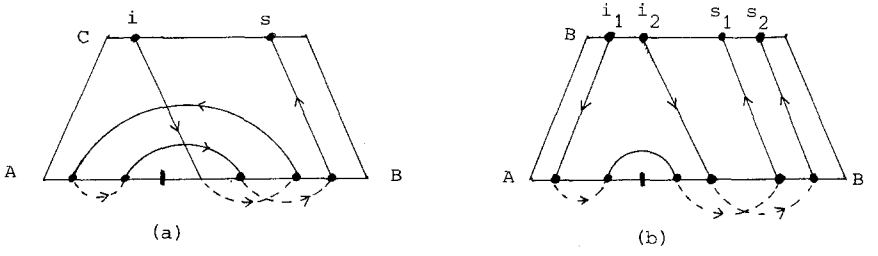

FIG. 7. Augmented dependency graphs.

all *is*-paths. Thus we construct an attribute grammar G' with a particular nonterminal Z, such that the R-language of G is finite if and only if $\{\#_R(\pi) | \pi$ is an *is*-path from $i_0(Z)$ to $s_0(Z)$ in $G'\}$ is finite, where i_0 and s_0 are new attributes. G' is constructed from G by adding attributes i_0 and s_0 to each nonterminal of G, and adding a new production $Z' \to Z$ where Z' is the (new) initial nonterminal of G' and Z the one of G. For each production $p: X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ of G the semantic rules are changed by adding $i_0(X_0)$ to each right-hand side of a semantic rule, i.e., each attribute defined by a semantic rule is made to depend on $i_0(X_0)$. Also semantic rules for i_0 and s_0 are added: $i_0(X_j) := i_0(X_0)$ for $1 \leqslant j \leqslant n_p$, and $s_0(X_0) := f(a_1, a_2, ..., a_m)$ where $a_1, ..., a_m$ consist of (i) all attributes which occur in the right-hand sides of the semantic rules of p in G, and (ii) all $s_0(X_j)$, $1 \leqslant j \leqslant n_p$. In the production $Z' \to Z$ (which is just there for completeness sake) $i_0(Z)$ is initialized to a constant. This ends the construction of G'. It should be clear that the size of G' is linear in the size of G. (We note here that defining $s_0(X_0)$ such that $a_1, ..., a_m$ are all *i*-attributes of the father X_0 and all *s*-attributes of the sons X_j, would give an easier construction, with $R(G) = R(Z)$, but would in general square the size of the attribute grammar.)

Let $R(Z) = \{\#_R(\pi) \mid \pi$ is an *is*-path from $i_0(Z)$ to $s_0(Z)\}$. We now want to show that $R(G)$ is finite if and only if $R(Z)$ is finite. Note first that none of the newly added dependency edges is an R-edge. Consider an *is*-path π' from $i_0(Z)$ to $s_0(Z)$ in G'; π' can clearly be decomposed into $\pi' = \pi_1 \cdot \pi \cdot \pi_2$, where π_1 consists entirely of new edges (each starting at some i_0), π is a path in G (i.e., consisting of old edges), and π_2 consists again of new edges (each leading to some s_0). This shows that $R(Z) \subseteq R(G)$. Now consider any dependency path π in G and assume that its last node does not have out-degree 0 in the dependency network (i.e., at least one edge is leaving it). Then there are π_1 and π_2 such that $\pi' = \pi_1 \cdot \pi \cdot \pi_2$ is an *is*-path from $i_0(Z)$ to $s_0(Z)$ in G'. From this it easily follows that $\{R^m \mid R^{m+1} \in R(G)\} \subseteq R(Z)$. Hence $\{R^m \mid R^{m+1} \in R(G)\} \subseteq R(Z) \subseteq R(G)$, and so $R(Z)$ is finite if and only if $R(G)$ is finite.

It now remains to decide whether $R(Z) = \{\#_R(\pi) \mid \pi$ is an *is*-path from $i_0(Z)$ to $s_0(Z)$ in $G'\}$ is finite. Using Lemma 4.2, we construct (in exponential time) the context-free grammar H corresponding to G' which generates all its acyclic *is*-paths. Then, we first test whether G' is circular (in time polynomial in the size of H) in the same way as in Jazayeri *et al.*, (1975): G' is circular if for some production $p: X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ of G' there is a cycle in the graph obtained by adding to $D(p)$ all edges (i, s) such that $(X_j, \{(i, s)\})$ generates some string in H. If G' is circular, then $R(Z)$ is infinite and G is not pure multi-pass. If G' is noncircular, then all *is*-paths are acyclic and so $R(Z) = \{R^m \mid (Z, \{(i_0, s_0)\}) \Rightarrow^* R^m$ in $H\}$. Since finiteness of a context-free language can be decided in polynomial time, we can decide whether $R(Z)$, and hence $R(G)$, is finite.

This shows that the pure multi-pass property can be decided in exponential time. ∎

Suppose that the decision method of Theorem 4.4 tells us that attribute grammar G is pure multi-pass. How many passes are actually needed to evaluate all attributes in all derivation trees of G? From Theorem 2.4 we know that the number of passes needed is one plus the length of the longest string in $R(G)$. Since the length of the longest string in a finite language generated by a context-free grammar is at most exponential in the size of the grammar (by the pumping lemma), the proof of Theorem 4.4 shows that an upper bound for the number of passes needed is $2^{2^{dn}}$ for some $d > 0$, where n is the size of G. We conjecture that this is the optimal upper bound. In any case there is an exponential lower bound. To show this consider, for each n, attribute grammar G_n with productions $Z \to A_0$, $A_0 \to A_1A_1$, $A_1 \to A_2A_2$,..., $A_{n-1} \to A_nA_n$, $A_n \to BB$, and $B \to \lambda$. Every nonterminal, except Z, has one i-attribute and one s-attribute. The dependency graphs of the productions are given in Fig. 8. Clearly G_n has one complete derivation tree t_n, which has 2^n nodes labeled A_n. Hence G_n is pure multi-pass. Since there is a dependency path in $D(t_n)$ with 2^n R-edges, Theorem 2.4 implies that the number of passes needed for G_n is $2^n + 1$, which is exponential in the size of G_n. We note that it is also unclear how much time is needed to compute, for a given pure multi-pass AG, the minimal number of passes needed. By an easy variation of Theorem 3.4(2) it can be shown that it takes at least exponential time; it can also be shown that double exponential time is an upper bound.

Since such a large number of passes is not attractive, it would be nice to be able to decide whether an AG is pure k-pass for a fixed (small) k. In the rest of this section we will show that this is possible, and even in polynomial time (where the degree of the polynomium depends of course on k). The main idea on which this is based is the following. From Theorem 2.4 we know that an attribute grammar G is pure k-pass if and only if $M(G) \leqslant k - 1$, where $M(G)$ is the maximal length of a string in $R(G)$. Since, clearly, $R(G)$ is substring-closed (i.e., if $R^m \in R(G)$ and $n < m$, then $R^n \in R(G)$), $M(G) \leqslant k - 1$ if and only if $R^k \notin R(G)$. Let us state this as a corollary (of Theorem 2.4).

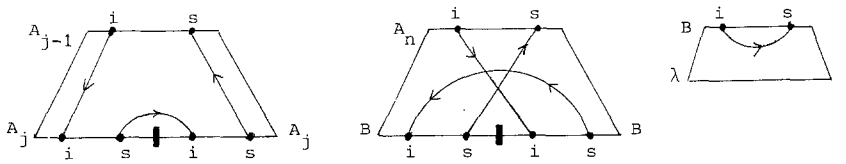

FIG. 8. Dependency graphs of G_n; $1 \leqslant j \leqslant n$.

4.5. COROLLARY. *An attribute grammar G is pure k-pass if and only if* $R^k \notin R(G)$. ∎

Thus, deciding the pure k-pass property amounts to checking the (non) existence of a dependency path π such that $\#_R(\pi) = k$. If one considers the way such a path "visits" a subtree t' of the derivation tree (cf. the discussion preceding Lemma 4.2), it should be clear that it can visit t' at most $k+1$ times: if an (acyclic) path enters and exists a node m times, then the path contains at least $m-1$ R-edges, see Fig. 9.

4.6. DEFINITION. For $m \geqslant 1$, a path π in a dependency network $D(t)$ of a derivation tree t is *m-visit* if, for every subtree t' of t, the *is*-graph D of X has at most m edges, where X is the root of t' and $D = \{(i_1, s_1) \mid \pi$ has a subpath which is an *is*-path from $i_1(X)$ to $s_1(X)$ in $D(t')\}$. ∎

Thus, since we only have to consider $(k+1)$-visit paths, the context-free grammar H, constructed according to Lemma 4.2, needs to contain only nonterminals (X, D) where the *is*-graph D has at most $k+1$ edges. Clearly, there are only polynomially many such D; to be precise at most $n^{2(k+1)}$ where n is the number of attributes of G. Therefore, assuming that the underlying context-free grammar of G is, e.g., in Chomsky Normal Form, the size of H is polynomial in the size of G (viz. n^{dk} for some $d > 0$). Hence the pure k-pass property can be decided in time n^{dk} for some $d > 0$, by the same method as for the pure multi-pass property, but restricted to $(k+1)$-visit paths (and deciding nonmembership of R^k rather than finiteness). Apart from technicalities such as dealing with circular AG (which are not pure k-pass) the only weakness in the above is that we do not yet know whether every AG can be transformed into Chomsky Normal Form (in polynomial time). In the next theorem we show a slightly weaker result, which is sufficient for our purpose.

4.7. DEFINITION. An attribute grammar G is in Almost Chomsky Normal Form if every production of the underlying context-free grammar has a right-hand side of length at most 2. ∎

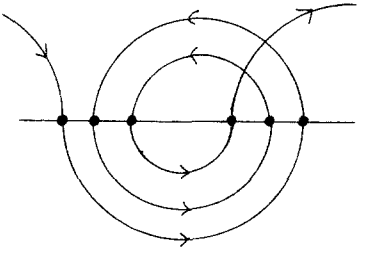

FIG. 9. Three visits, two R-edges.

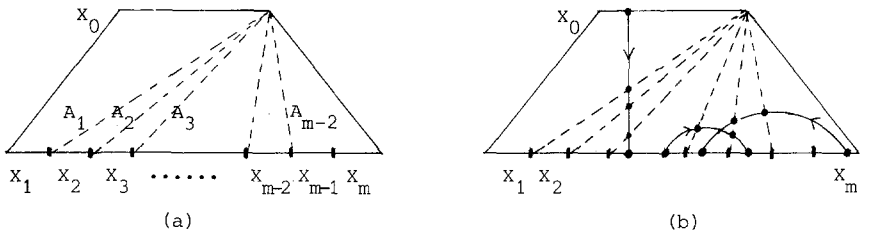

FIG. 10. Dependency graph cut into pieces.

4.8. THEOREM. *For every attribute grammar G there is an attribute grammar G' in Almost Chomsky Normal Form such that*

(i) *G' is equivalent to G in the sense that it assigns the same meaning as G to every string (in particular, they have the same underlying context-free language);*

(ii) *G' uses the same semantic rules as G, apart from the names of attributes, and some additional semantic rules of the form $a := b$, where a and b are attributes;*

(iii) *for every $k \geqslant 1$, G' is pure k-pass if and only if G is pure k-pass. Moreover, G' can be constructed from G in time $O(n^2 \log n)$ where n is the size of G.*

Proof. Consider a production $p: X_0 \to X_1 X_2 \cdots X_m$ of G with $m > 2$, where X_j is a terminal or a nonterminal for $j \geqslant 1$. We take the usual construction to simulate p by short productions: new nonterminals $A_1, A_2, \ldots, A_{m-2}$ are introduced and p is replaced by all productions $X_0 \to X_1 A_1$, $A_1 \to X_2 A_2, \ldots, A_{m-3} \to X_{m-2} A_{m-2}$, and $A_{m-2} \to X_{m-1} X_m$. The new nonterminals have to have attributes which pass the appropriate attribute values of the old nonterminals. Consider Fig. 10a. Pictorially speaking, production p is cut into the $m-1$ small productions along the dotted lines, where the jth dotted line corresponds to A_j. Similarly, $D(p)$ is cut into $m-1$ small dependency graphs, as in Fig. 10b. A_j will have an attribute corresponding to each intersection of its dotted line with a dependency edge (an i-attribute if the direction of the edge is from left-to-right, and an s-attribute if it is an R-edge). In this way each dependency edge is cut into at most $m-1$ new dependency edges.

Formally, in G', A_j has an i-attribute $\langle a, X_k \rangle$ if $a(X_k)$ is used for some $b(X_l)$, $0 \leqslant k \leqslant j < l \leqslant m$, and A_j has an s-attribute $\langle a, X_l \rangle$ if $a(X_l)$ is used for some $b(X_k)$, $0 \leqslant k \leqslant j < l \leqslant m$. The semantic rules of G' are obtained by changes in the rules of G and adding new "passing" rules, as follows.

$X_0 \to X_1 A_1$

$S(X_0)$, $I(X_1)$: replace $a(X_l)$ by $\langle a, X_l\rangle(A_1)$ for $l > 1$

$I(A_1)$: $\langle a, X_k\rangle(A_1) := a(X_k)$ for $k \leqslant 1$,

$A_{j-1} \to X_j A_j$

$I(X_j)$: replace $a(X_k)$ by $\langle a, X_k\rangle(A_{j-1})$ for $k < j$, and
replace $a(X_l)$ by $\langle a, X_l\rangle(A_j)$ for $l > j$,

$S(A_{j-1})$: $\langle a, X_l\rangle(A_{j-1}) := a(X_l)$ if $l = j$

$:= \langle a, X_l\rangle(A_j)$ if $l > j$,

$I(A_j)$: $\langle a, X_k\rangle(A_j) := a(X_k)$ if $k = j$

$:= \langle a, X_k\rangle(A_{j-1})$ if $k < j$,

$A_{m-2} \to X_{m-1} X_m$

$I(X_{m-1})$, $I(X_m)$: replace $a(X_k)$ by $\langle a, X_k\rangle(A_{m-2})$ for $k < m-1$

$S(A_{m-2})$: $\langle a, X_l\rangle(A_{m-2}) := a(X_l)$ for $l \geqslant m-1$.

Hence every semantic rule $a := f(a_1, \ldots, a_r)$ is replaced by a semantic rule $a := f(a'_1, \ldots, a'_r)$ and at most $m-2$ "A-rules" for each a'_j, $1 \leqslant j \leqslant r$. The size of the A-rules is therefore proportional to $m-2$ times the size of $a := f(a'_1, \ldots, a'_r)$. Since to write down A_j or one of its attributes takes space $O(\log m)$, the size of G' is $O(n(m-1)\log m)$ and hence $O(n^2 \log n)$, where n is the size of G.

Each edge is cut into at most $m-2$ pieces; in case of an R-edge only the last piece is an R-edge; in all other cases no new R-edges are introduced. Since clearly (pictorially speaking) G and G' have the "same" dependency networks (apart from the introduction of new intermediate nodes), this shows that G and G' have the same R-language. Hence, by Theorem 2.4, the number of passes is preserved.

It should be clear that G and G' are equivalent. ∎

To show formally the decidability of the pure k-pass property we first need the analogue of Lemma 4.2.

4.9. LEMMA. *For every attribute grammar G in Almost Chomsky Normal Form and every integer $k \geqslant 0$, a context-free grammar H can be constructed in time n^{dk} for some $d > 0$ (where n is the size of G) such that*

(i) *H has nonterminals of the form (X, D) where X is a nonterminal of G and D is an is-graph of X with at most $k+1$ edges; the terminal alphabet of H is $\{R\}$;*

(ii) *for every nonterminal X_0 of G and attributes $i_0(X_0)$ and $s_0(X_0)$,*

$(X_0, \{(i_0, s_0)\}) \Rightarrow^* R^m$ if and only if $\#_R(\pi) = m$ for some acyclic $(k+1)$-visit is-path π from $i_0(X_0)$ to $s_0(X_0)$.

Moreover, d does not depend on k or n.

Proof. The proof is entirely analogous to the one of Lemma 4.2, restricting attention to $(k+1)$-visit paths. In the actual construction of H (see to the end of the proof of Lemma 4.2) we only have to consider sets $\{\pi_1, ..., \pi_v\}$ of paths in the augmented dependency path of a production p of G such that the total number of attributes in $\pi_1, ..., \pi_v$ is at most $6(k+1)$. This follows from the fact that G is in Almost Chomsky Normal Form and so there can be at most $2(k+1)$ augmented edges in the path; moreover $v \leqslant k+1$, and so the number of attributes is at most $2 \cdot 2(k+1) + 2v \leqslant 6(k+1)$. Since the number of attributes is at most n (the size of G), at most $n^{6(k+1)}$ productions correspond to each production p of G. Consequently, since each such production of H can be written down in at most three times the space of production p with its semantic rules (cf. the proof of Lemma 4.2), the size of H is at most $n^{6(k+1)} \cdot 3n = 3n^{6k+7}$. Hence H can be constructed in time n^{dk} for some $d > 0$. ∎

Finally we prove that the pure k-pass property is decidable in polynomial time.

4.10. THEOREM. *It is decidable for an arbitrary attribute grammar G and an arbitrary integer $k \geqslant 1$, whether G is pure k-pass, in time n^{dk} for some $d > 0$ (where n is the size of G, and d does not depend on n or k).*

Proof. By Theorem 4.8 we may assume that G is in Almost Chomsky Normal Form and so Lemma 4.9 is applicable.

Let $R_k(G) = \{R^m \mid m = \#_R(\pi)$ for some $(k+1)$-visit dependency path of $G\}$. As argued before, G is pure k-pass if and only if $R^k \notin R_k(G)$, cf. Corollary 4.5 and the discussion following it.

We now first apply the same transformation to G as in the proof of Theorem 4.4, resulting in an attribute grammar G' with nonterminal Z and attributes i_0 and s_0. The only difference is that this time we want $R(Z) = R(G)$ precisely. To obtain this, the rules for $s_0(X_0)$ become $s_0(X_0) := f(a_1, ..., a_m)$ where $a_1, ..., a_m$ are all i-attributes of the father X_0 and all s-attributes (including s_0) of the sons X_j. The size of G' is then $O(n^2)$ where n is the size of G. Note that $R(G') = R(Z) = R(G)$. Denote by $R_k(Z)$ the set $\{\#_R(\pi) \mid \pi$ is a $(k+1)$-visit *is*-path from $i_0(Z)$ to $s_0(Z)$ in $G'\}$. It should be clear (from the proof of Theorem 4.4) that $R_k(Z) = R_k(G)$.

It now remains to decide whether $R^k \in R_k(Z)$. By Lemma 4.9 we construct (in time n^{dk}) the context-free grammar H corresponding to G' which generates all its acyclic $(k+1)$-visit paths. The next step is to test circularity, as in the proof of Theorem 4.4. However, due to the fact that

only $(k+1)$-visit paths are considered, a restricted kind of circularity is tested; let us call it "k-circularity". If G' is k-circular, then it is certainly circular and hence not pure k-pass. If G' is not k-circular, then we claim that $R^k \in R_k(Z)$ if and only if $(Z, \{(i_0, s_0)\}) \Rightarrow^* R^k$. Using this claim we can test whether R^k is in the context-free language generated by $(Z, \{(i_0, s_0)\})$ in time polynomial in k and the size of H, and thus decide whether G is pure k-pass in time n^{dk} for some $d > 0$. To prove the claim, note first that the if-direction is trivial by Lemma 4.9. For the only-if-direction, assume that π is a cyclic $(k+1)$-visit *is*-path from $i_0(Z)$ to $s_0(Z)$ such that $\#_R(\pi) = k$. Can we find an acyclic path with the same property? The answer is yes: one of π's subpaths can be taken. In fact, if not, then all of π's acyclic subpaths contain less than k R-edges, and consequently π would be detected by the k-circularity test. ∎

Note that after discovering that an AG is pure k-pass it can easily be turned into an equivalent simple k-pass AG by Theorem 2.6.

The results of this section can also be proved for the pure alternating multi-pass AG, cf. the discussion following Theorem 3.4. Instead of the number of R-edges one should count the number of alternations between R-edges and "L-edges" along a dependency path; the proofs become more complicated, but the ideas are the same.

5. Path Languages

The approach to the pure multi-pass problem (and the noncircularity problem in Jazayeri *et al.* (1975)) is a "path-approach": all dependency paths of an attribute grammar G are considered and the number of R-edges in each path is counted. This can be done by somehow simulating dependency paths by the derivations of a context-free grammar (cf. Lemma 4.2, and the construction in Jazayeri *et al.* (1975)). In this section we take a more formal point of view and try to extract the general idea from this technique. Each dependency path can be viewed as a string over the alphabet E of dependency edges, and consequently the set of all dependency paths is a formal language over the alphabet E. Note that the R-language of G (Definition 4.1) is a homomorphic image of this "dependency path language" (by the homomorphism h such that, for any edge $e \in E$, $h(e) = R$ if e is an R-edge, $h(e) = \lambda$ otherwise). Thus path properties of the grammar G can be translated into properties of its dependency path language. Hence it would be nice to know by what kind of formal device the dependency path language of an attribute grammar can in general be generated. Properties of the dependency path language then translate in their turn into properties of the formal device. In this section we show that such a formal device is the top-down tree-to-string transducer (Engelfriet *et al.*, 1980): it can translate

each derivation tree nondeterministically into every dependency path of the tree. More precisely, the appropriate device is the "finite-copying" top-down tree-to-string transducer. As will be shown, not only is each dependency path language of an attribute grammar the output language of a finite-copying top-down tree-to-string transducer, but, vice versa, each such output language is very close to the dependency path language of an attribute grammar (in fact it is its image under a homomorphism).

We need some terminology on top-down tree-to-string transducers (which are just the usual top-down tree transducers, of which the yield of the output tree is taken). For more precise definitions, see, e.g., Engelfriet *et al.* (1980). A *top-down tree-to-string transducer* (abbreviated yT-transducer, where y stands for yield) is a device $M = (Q, \Sigma, \Delta, q_0, R)$, where Q is a finite set of states, Σ is the (ranked) input alphabet, Δ is the output alphabet, $q_0 \in Q$ is the initial state, and R is the finite set of rules of the form

$$(*)\quad q(\sigma(x_1 \cdots x_n)) \to w_0 q_1(x_{i_1})\, w_1 q_2(x_{i_2})\, w_2 \cdots q_r(x_{i_r})\, w_r,$$

where $n, r \geqslant 0$; $q, q_1, \ldots, q_r \in Q$; $\sigma \in \Sigma$ (and σ has rank n); $w_0, w_1, \ldots, w_r \in \Delta^*$, and $1 \leqslant i_m \leqslant n$ for $1 \leqslant m \leqslant r$.

Intuitively rule (*) says that the q-translation of a tree $\sigma(t_1 \cdots t_n)$ with root labeled σ and direct subtrees $t_1, \ldots, t_n$, is (nondeterministically) the concatenation of strings $w_0, v_1, w_1, v_2, w_2, \ldots, v_r, w_r$ where v_m is a q_m-translation of t_{i_m} $(1 \leqslant m \leqslant r)$. Formally, we define $q(\sigma(t_1 \cdots t_n)) \Rightarrow^* w_0 q_1(t_{i_1})\, w_1 \cdots q_r(t_{i_r})\, w_r$ and extend this in the usual way to a derivation relation $\Rightarrow^*$. Thus, if $q_m(t_{i_m}) \Rightarrow^* v_m$, then $q(\sigma(t_1 \cdots t_n)) \Rightarrow^* w_0 v_1 w_1 v_2 w_2 \cdots v_r w_r$. The translation realized by M is the relation $\{\langle t, w\rangle \mid t$ is a tree over Σ, $w \in \Delta^*$, and $q_0(t) \Rightarrow^* w\}$; it is also denoted by M.

M is *deterministic* (yDT) if different rules in R have different left-hand sides. For $k \geqslant 1$, M is *k-copying* ($yT_{\mathrm{fc}(k)}$) if the following holds for every input tree t and (occurrence of a) subtree t' of t: if $q_0(t) \Rightarrow^* w_0 q_1(t')\, w_1 q_2(t')\, w_2 \cdots q_m(t')\, w_m$, where $q_j \in Q$ and $w_j \in \Delta^*$, then $m \leqslant k$. M is *finite-copying* (yT_{fc}) if it is k-copying for some $k \geqslant 1$. Thus, intuitively, M is finite-copying if there is a bound on the number of occurrences of translations of a subtree in the translation of the input tree. The intuitive reason that the yT-transducer generating dependency paths will be finite-copying is that every dependency path of an attribute grammar is k-visit (where k is, roughly, the number of attributes in the grammar).

We will be interested in the output language of a (finite-copying) yT-transducer, when the set of derivation trees of a context-free grammar (the underlying context-free grammar of the AG) is given to it as input language. It is usual to allow arbitrary recognizable tree languages (RECOG, see, e.g., Thatcher (1973)) as input languages. Thus we define $yT_{\mathrm{fc}}(\mathrm{RECOG})$ to be the class of languages $M(L) = \{w \in \Delta^* \mid q_0(t) \Rightarrow^* w$ for some $t \in L\}$, where

M is a yT_{fc}-transducer and L is a recognizable tree language; and similarly for other classes of transducers. Since every recognizable tree language is a projection of the set of derivation trees of a context-free grammar, we obtain precisely the desired class of output languages.

We now turn to the proof that every dependency path language can be generated by a yT_{fc}-transducer. We first prove an analogue of Lemma 4.2, restricting attention to *is*-paths. Intuitively, since we have to generate an actual dependency path and not just its R-edges (as in Section 4), we cannot use a context-free grammar any more. Actually, the context-free grammar H constructed in Lemma 4.2 generates all pieces of a path π through some subtree t' of the derivation tree simultaneously, thus destroying the correct order of the edges of π. Clearly, using a context-free grammar, we cannot generate these pieces at their correct positions, because it would not be guaranteed any more that they run through the same subtree t'. However, passing the subtree t' as a parameter to each of the pieces, as can be done in a yT-transducer, it is again possible to generate the correct path.

In the next lemma, when considering derivation trees as an input to the yT-transducer, we will assume that each node x labeled X_0 is actually labeled p, where $p\colon X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ is the production applied at x; moreover we assume that x has n_p sons, i.e., we disregard all terminals. Thus we identify two different (but closely related) types of derivation tree. Note that the relationship between the two can be established by a finite tree automaton and so the input tree language to the tree transducer is recognizable. For the notion of a k-visit path see Definition 4.6.

5.1. LEMMA. *For every noncircular attribute grammar G there is a (nondeterministic) top-down tree-to-string transducer M such that*

(i) *M has states of the form $\langle i, s\rangle$ where i and s are an i-attribute and an s-attribute of some nonterminal of G, respectively; the output alphabet of M is the set E of dependency edges in the dependency graphs of productions of G;*

(ii) *for every derivation tree t of G and attributes i_0 and s_0 of the root of t, $\langle i_0, s_0\rangle(t) \Rightarrow^* w$ if and only if $w \in E^*$ is an is-path from i_0 to s_0 in $D(t)$. Moreover, if every is-path of G is k-visit, then M is k-copying. In particular, M is k-copying where k is the maximum of $\min(\#I(X), \#S(X))$ for all nonterminals X of G ($\#$ denotes cardinality).*

Proof. The construction is similar to but easier than the one in Lemma 4.2.

Choose a production $p\colon X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ of G, choose attributes $i_0(X_0)$ and $s_0(X_0)$, and choose a path π in the augmented dependency graph of p (cf. the proof of Lemma 4.2) from $i_0(X_0)$ to $s_0(X_0)$. Clearly π consists of

a sequence $a_1, a_2, \ldots, a_{2u-1}, a_{2u}$ of attributes ($u \geqslant 1$) such that (a_{2j-1}, a_{2j}) is a dependency edge and (a_{2j}, a_{2j+1}) is an augmented edge. Construct rule $\langle i_0, s_0\rangle(p(x_1 \cdots x_{n_p})) \to e_1 e_2 \cdots e_{2u-1}$ of M, where e_{2j-1} is the dependency edge $(a_{2j-1}, a_{2j}) \in E$, and $e_{2j} = \langle a_{2j}, a_{2j+1}\rangle(x_k)$ if a_{2j} and a_{2j+1} are attributes of X_k.

It is easy to prove that M satisfies (ii) by induction on the height of t. Moreover, it is also not difficult to see (as in Lemma 4.2) that if $\langle i_0, s_0\rangle(t) \Rightarrow^* w_0\langle i_1, s_1\rangle(t')\, w_1\langle i_2, s_2\rangle(t')\, w_2 \cdots \langle i_m, s_m\rangle(t')\, w_m$ where t' is (an occurrence of) a subtree of t and $w_j \in E^*$, then there is an *is*-path π from i_0 to s_0 in $D(t)$ such that π has m subpaths which are *is*-paths in $D(t')$ from $i_j(X)$ to $s_j(X)$ where X is the root of t'. Hence, if π is k-visit, then $m \leqslant k$. Consequently, if every path π is k-visit, then M is k-copying. ∎

5.2. EXAMPLE. Consider again the AG G_0 of Examples 2.5 and 4.3. Give names to the dependency edges such that the dependency path in Fig. 2a is *pqrtuvwrxyzab*. Number the productions $p_1: C \to AB$, $p_2: B \to AB$, $p_3: A \to a$, and $p_4: B \to b$. Then at least the following rules belong to the yT-transducer M constructed in Lemma 5.1.

$$\langle i, s\rangle(p_1(x_1x_2)) \to p\langle i_1, s_1\rangle(x_2)\, w\langle i, s\rangle(x_1)\, x\langle i_2, s_2\rangle(x_2)\, b,$$
$$\langle i_1, s_1\rangle(p_2(x_1x_2)) \to q\langle i, s\rangle(x_1)\, t\langle i_1, s_1\rangle(x_2)\, v,$$
$$\langle i_2, s_2\rangle(p_2(x_1x_2)) \to y\langle i_2, s_2\rangle(x_2)\, a,$$
$$\langle i, s\rangle(p_3) \to r,$$
$$\langle i_1, s_1\rangle(p_4) \to u,$$
$$\langle i_2, s_2\rangle(p_4) \to z.$$

The set of *is*-paths from $i(C)$ to $s(C)$ is $\{p(qrt)^n uv^n wrxy^n za^n b \mid n \geqslant 0\}$, where n corresponds to the number of times p_2 is used. Note that this is not a context-free language. It is easy to see that these strings are generated by the above transduction rules starting with $\langle i, s\rangle(t_1)$ for all derivation trees t_1 with root C. ∎

Lemma 5.1 can now be used to show that the dependency path language of an attribute grammar is in yT_{fc}(RECOG). In the proof we use the obvious fact that yT_{fc}(RECOG) and $yT_{\mathrm{fc}(k)}$(RECOG) are closed under homomorphisms. Actually they are full substitution-closed AFLs (Engelfriet *et al.*, 1980).

5.3. DEFINITION. The *dependency path language* of an AG G, denoted by $dpl(G)$, is the language $\{\pi \in E^* \mid \pi$ is a path in $D(t)$ for some complete derivation tree t of $G\}$, where E is the set of edges in the dependency graphs of productions of G. ∎

5.4. THEOREM. *For every noncircular attribute grammar* G, $dpl(G) \in yT_{fc}(RECOG)$.

Proof. By Lemma 5.1 we know that, for every AG, the language of *is*-paths belongs to yT_{fc}(RECOG). We use the trick in the beginning of the proof of Theorem 4.4 (or better Theorem 4.10) to transform G into G' such that the *is*-paths of G' are of the form $\pi_1 \cdot \pi \cdot \pi_2$, where $\pi \in dpl(G)$ and π_1, π_2 consist entirely of new edges (and every $\pi \in dpl(G)$ appears in an *is*-path of G'). Hence $dpl(G)$ is obtained from the set of *is*-paths of G' by the homomorphism which erases all new edges (and is the identity on the edges of G). Since yT_{fc}(RECOG) is closed under homomorphisms, this proves the theorem. ∎

The decidability of the pure multi-pass property could have been shown using this result, as follows. Clearly, for every AG G, $R(G) = h(dpl(G))$, where h is the homomorphism such that $h(e) = R$ if e is an R-edge and $h(e) = \lambda$ otherwise. Since yT_{fc}(RECOG) is closed under homomorphisms, $R(G) \in yT_{fc}$(RECOG). However, it is known (Engelfriet *et al.*, 1980) that every language in yT_{fc}(RECOG) over a one-letter alphabet is actually context-free. Thus $R(G)$ is context-free, as also shown in the previous section. The reader might be interested in the construction used in the proof of Theorem 3.2.6 of Engelfriet *et al.* (1980) to see the connection between Lemmas 4.2 and 5.1.

To show the similarity to the constructions in Section 4, the proof of Theorem 5.4 has been given by associating a yT-transducer with each dependency path language. Actually an easier proof is possible. As shown in Engelfriet *et al.* (1980) yT_{fc}(RECOG) equals the class of output languages of the deterministic tree-walking automaton of Aho and Ullman (1971). Now obviously, a nondeterministic tree-walking automaton can be constructed which, on a derivation tree t of an AG, follows and outputs a dependency path through $D(t)$. Moreover, since every path is k-visit (where k depends on G only), the automaton is "finite-visit" (or "finite-crossing") and can hence be transformed into a deterministic tree-walking automaton by a change of the input language (Engelfriet *et al.*, 1980).

In the rest of this section we consider attribute grammars whose attributes have trees as values. Viewing the derivation tree as an input tree and the value of the designated attribute of its root as the output tree, the attribute grammar is thus turned into a tree transducer (or, viewing the yield of the derivation tree as input string, into a string-to-tree transducer), cf. (Engelfriet, 1980; Engelfriet and Filè, 1979). The motivation to consider such attribute grammars is that trees can be viewed as formal, uninterpreted expressions, a symbol of rank n standing for an operation with n arguments. Taking trees as attribute values therefore amounts to viewing attribute grammars as program schemes which assign an uninterpreted meaning (i.e.,

an expression) to each string of the underlying context-free language. Comparing the classes of string-to-tree translations defined by different classes of attribute grammars thus gives insight into the power of the features used in these classes, independent of the semantics, i.e., the actual meaning of the operations in the semantic rules. As an example, note that it is shown in Theorem 2.6 that the pure and simple multi-pass AG have the same power; similarly, in Theorem 4.8 it is shown that the power of multi-pass AG stays the same when only AG in Almost Chomsky Normal Form are allowed. To compare classes of translations of AG we will in particular consider the classes of output tree languages of AG, cf. (Engelfriet and Filè, 1979). To compare classes of tree languages one can often use the "path-approach," i.e., consider all paths from root to leaf in the trees of the language, and compare these classes of path languages instead (see, e.g., (Rounds, 1970b; Engelfriet and Slutzki, 1979)). We will show that these path languages of attribute grammars are closely related to the dependency path languages and, in fact, the class of all path languages of output tree languages of attribute grammars is equal to $yT_{\mathrm{fc}}(\mathrm{RECOG})$.

We need some terminology on trees. A *ranked alphabet* Σ is an alphabet such that every symbol $\sigma \in \Sigma$ has a finite number of nonnegative ranks (intuitively, its possible number of sons or arguments). Σ is *monadic* if every symbol has ranks 1 and 0. A *tree* over Σ is either a symbol $\sigma \in \Sigma$ of rank 0 or it is a string $\sigma(t_1 t_2 \cdots t_n)$, where $\sigma \in \Sigma$ has rank n and $t_1, t_2, \ldots, t_n$ are trees over Σ. T_Σ denotes the set of all trees over Σ. The set of *variables* is $\{x_1, x_2, \ldots\}$. $T_\Sigma[x_1, x_2, \ldots, x_n]$ denotes the set of all trees over $\Sigma \cup \{x_1, \ldots, x_n\}$, where each variable has rank 0. For a tree $t \in T_\Sigma[x_1, \ldots, x_n]$ and strings $s_1, \ldots, s_n$, $t[s_1, \ldots, s_n]$ denotes the result of substituting s_i for x_i everywhere in t, $1 \leqslant i \leqslant n$. Note that if $s_1, \ldots, s_n \in T_\Sigma$, then $t[s_1, \ldots, s_n] \in T_\Sigma$.

An attribute grammar G is *tree-valued* if all its attribute values are trees over a given ranked output alphabet Σ and all its semantic rules are of the form $a_0 := t[a_1, \ldots, a_n]$, where $t \in T_\Sigma[x_1, \ldots, x_n]$ and $a_0, a_1, \ldots, a_n$ are appropriate attributes of some production. Evaluation of such a rule consists of substituting the value t_i (of a_i) for a_i everywhere in $t[a_1, \ldots, a_n]$, i.e., if t_0 denotes the value of a_0, then $t_0 = t[t_1, \ldots, t_n]$. Hence all attribute values are in T_Σ. It should be clear that on the one hand G is an ordinary AG with trees as values and operations on trees used in the semantic rules (in particular, the operation $t_1, \ldots, t_n \mapsto \sigma(t_1 \cdots t_n)$ for every σ), and on the other hand G may be viewed as a "schematic" attribute grammar whose trees represent expressions to be interpreted in any suitable domain.

For an AG G, we denote by $\mathrm{OUT}(G)$ the set of all meanings of complete derivation trees, i.e., for each complete derivation tree t the value of the designated attribute of the root of t. Thus, $\mathrm{OUT}(G)$ is the range of the string-to-value (or, tree-to-value) translation of G. Note that for a tree-valued AG G, $\mathrm{OUT}(G)$ is a tree language. By $\mathrm{OUT}(\mathrm{AG}, \mathrm{TREES})$ we denote the class of

all tree languages OUT(G) of tree-valued attribute grammars G; and similarly for OUT (X-AG, TREES), where X-AG is any class of attribute grammars.

For a ranked alphabet Σ, the *path-alphabet* of Σ, denoted $\pi(\Sigma)$, is the set $\{\sigma_1 \mid \sigma \in \Sigma \text{ has rank } 0\} \cup \{\sigma_j \mid \sigma \in \Sigma \text{ and } 1 \leqslant j \leqslant n \text{ where } n \text{ is the rank of } \sigma\}$. Intuitively, σ_j denotes the edge from a node labeled σ to its jth son. For σ of rank 0, σ_1 means that the path ends at a node labeled σ (the index 1 is a trick to be able to identify trees over a monadic alphabet with strings over that alphabet). For a tree t over Σ, $\pi(t)$ denotes the set of all paths from the root of t to some leaf of t. Formally, if σ has rank 0, then $\pi(\sigma) = \{\sigma_1\}$, and if σ has rank n, then $\pi(\sigma(t_1 \cdots t_n)) = \bigcup\{\sigma_j \cdot \pi(t_j) \mid 1 \leqslant j \leqslant n\}$. For a tree language L, $\pi(L) = \bigcup\{\pi(t) \mid t \in L\}$ is the *path-language* corresponding to L; $\pi(L)$ is a language over the path-alphabet $\pi(\Sigma)$. Finally, for a class X of tree languages, $\pi(X) = \{\pi(L) \mid L \in X\}$ is the corresponding class of path languages.

In what follows we want to show that $\pi(\text{OUT(AG, TREES)}) = yT_{\text{fc}}(\text{RECOG})$. Paths through the output tree of some derivation tree t are very close to those paths in $D(t)$ which end at the designated s-attribute of the root of t and start at a node of $D(t)$ with in-degree 0, i.e., no incoming edges. To obtain a better correspondence we add to the dependency graph of a production $p: X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ a loose edge (with no start node) to a node in $I(X_j)$, $1 \leqslant j \leqslant n$, or in $S(X_0)$, whenever it is defined by a constant (i.e., depends on no attributes). In fact this is a natural extension of the concept of dependency graph which makes the graphs even more understandable.

5.5. DEFINITION. A dependency path in a dependency network is *complete* if it starts with a loose edge (as explained above) and ends at the designated s-attribute of the root. The *complete dependency path language* of an attribute grammar G, denoted cdpl(G), is the language $\{\pi \in E^* \mid \pi \text{ is a complete dependency path}\}$. ∎

5.6. THEOREM. *For every noncircular AG G,* cdpl(G) $\in yT_{\text{fc}}$(RECOG).

Proof. To use Lemma 5.1, we first transform G by the usual transformation into G'. This time we add an i-attribute i_0 to every nonterminal of G and for each production $p: X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ we add the semantic rules $i_0(X_j) := i_0(X_0)$, giving rise to a new edge in the dependency graph of p, and we change every constant semantic rule $a(X) := c$ into $a(X) := i_0(X_0)$, thus connecting the loose edge with end-node $a(X)$ to the start-node $i_0(X_0)$. It should be clear that cdpl(G) is obtained from the set of is-paths that run from $i_0(Z)$ to $s(Z)$, where Z is the initial nonterminal of G and s its designated attribute, by the homomorphism which erases all new edges. ∎

We finally prove the result on output path languages, and simultaneously link them more precisely to complete dependency path languages. Let HOM(CDPL(AG)) denote the class of all homomorphic images of complete dependency path languages of noncircular attribute grammars, i.e., HOM(CDPL(AG)) = $\{h(\text{cdpl}(G)) \mid h$ is a homomorphism, G is a noncircular attribute grammar$\}$. Note that this class is independent of whether the AG are tree-valued or not.

5.7. THEOREM.

$$\pi(OUT(AG, TREES)) = HOM(CDPL(AG)) = yT_{fc}(RECOG).$$

Proof. First we show the inclusions of π(OUT(AG, TREES)) and HOM(CDPL(AG)) in yT_{fc}(RECOG). By Theorem 5.6 CDPL(AG) $\subseteq$ yT_{fc}(RECOG), and since yT_{fc}(RECOG) is closed under homomorphisms, HOM(CDPL(AG)) $\subseteq yT_{fc}$(RECOG). To show that π(OUT(AG, TREES)) $\subseteq$ yT_{fc}(RECOG), let G be a tree-valued AG and let t be a complete derivation tree of G. Intuitively the output tree t' (i.e., the value of the designated attribute of the root of t) can be obtained by taking the dependency network $D(t)$ of t (which is a directed acyclic graph), unraveling it in the usual way into a tree (with the designated attribute as root), and finally applying to each node of this tree the semantic rule which defines it, in the sense that if for node a_0 the semantic rule is $a_0 := t[a_1,..., a_n]$, then the tree homomorphism h is applied which replaces the tree $a_0(t_1 \cdots t_n)$ by $t_0[h(t_1),..., h(t_n)]$; for the notion of tree homomorphism see, e.g., Engelfriet (1975). Thus the output tree t' is a tree-homomorphic image of the unraveled dependency network of t. With respect to paths the tree homomorphism can be replaced by a (nondeterministic) *gsm*: $\pi(t')$ is the image of the reverse of $cdpl(t)$ under a *gsm*-mapping g, where cdpl(t) is the set of all complete dependency paths in $D(t)$. Note that the reverse is taken because complete dependency paths end at the root, whereas paths in a tree start at the root. The *gsm*-mapping g is defined as follows. It has two states q_0 and q_∞, where q_0 is the initial state (in which g mostly is) and q_∞ is the final state (in which g halts). Suppose g reads in state q_0 the edge (a_0, a_j) of a reversed complete dependency path, and let a_0 be defined by the semantic rule $a_0 := t_0[a_1,..., a_n]$, and $1 \leqslant j \leqslant n$. Then g nondeterministically either outputs some path from the root of t_0 to an occurrence of x_j in t_0 (not including x_j itself) and continues in state q_0, or outputs a path from the root of t_0 to a (nonvariable) leaf of t_0 and halts in state q_∞ (or, if you wish, it continues in q_∞ to the end of the input, with output λ).

Hence $\pi(\text{OUT}(G)) = g(\text{cdpl}(G)^R)$, where R denotes the reversing operation on strings. Since, by Theorem 5.6, cdpl(G) $\in yT_{fc}$(RECOG), and since

yT_{fc}(RECOG) is closed under reverse and *gsm* mappings Engelfriet *et al.* (1980), the inclusion follows.

Now we show the inclusion of yT_{fc}(RECOG) in both HOM(CDPL(AG)) and π(OUT(AG, TREES)). Let $M = (Q, \Sigma, \Delta, q_0, R)$ be a k-copying yT-transducer and L a tree language in RECOG. We assume that $\lambda \notin M(L)$: the equality with π(OUT(AG, TREES)) only holds modulo λ because path languages do not contain λ. It is either not difficult to prove or follows directly from results in Engelfriet *et al.* (1980) that we may assume the following about M and L (for point (iii) note that we may assume that all strings of $M(L)$ start with the same symbol a, because yT_{fc}(RECOG) is closed under intersection with regular languages and the other two classes are obviously closed under union).

(i) M is deterministic (Lemma 3.2.3 of Engelfriet *et al.* (1980)).

(ii) L is the set of derivation trees of a context-free grammar G.

(iii) For the initial nonterminal Z of G there is exactly one rule in M; it has the form $q_0(Z(x_1)) \to aq(x_1)$ for some $a \in \Delta$ and $q \in Q$.

(iv) For every nonterminal X of G there is a unique sequence $\langle q_1, \ldots, q_m \rangle$ of *different* states of M ($m \leqslant k$) such that for every complete derivation tree t of G and for every subtree t' of t with root X, there occur exactly m translations of t' in the q_0-translation of t, such that the jth translation is a q_j-translation of t' ($1 \leqslant j \leqslant m$).

We turn G into an attribute grammar as follows. Z has the designated s-attribute s; if the sequence $\langle q_1, \ldots, q_m \rangle$ is associated with X by (iv), then X has i-attributes $iq_1, \ldots, iq_m$ and s-attributes $sq_1, \ldots, sq_m$. G is a tree-valued AG with output alphabet Δ, where each symbol in Δ has ranks 1 and 0 (i.e., it is a monadic alphabet). To describe the semantic rules of G we use $v[x_1]$ to denote the monadic tree $a_n(\cdots a_2(a_1(x_1))\cdots) \in T_\Delta[x_1]$, if $v = a_1 a_2 \cdots a_n$ and $a_j \in \Delta$ (for $v = \lambda$, $v[x_1]$ denotes x_1).

Consider a production p: $X_0 \to w_0 X_1 w_1 \cdots X_{n_p} w_{n_p}$ of G with $X_0 \neq Z$ and consider for each state q in the sequence associated with X_0 by (iv) the (unique) rule $q(X_0(x_1 \cdots x_{n_p})) \to v_0 q_1(x_{j_1}) v_1 q_2(x_{j_2}) v_2 \cdots q_r(x_{j_r}) v_r$ of M. For $r \geqslant 1$, the semantic rules associated to p are

$$\begin{aligned} iq_1(X_{j_1}) &:= v_0[iq(X_0)], \\ iq_{u+1}(X_{j_{u+1}}) &:= v_u[sq_u(X_{j_u})] \qquad \text{for} \quad 1 \leqslant u \leqslant r-1, \\ sq(X_0) &:= v_r[sq_r(X_{j_r})]. \end{aligned}$$

For $r = 0$, the semantic rule is $sq(X_0) := v_0[iq(X_0)]$. It should be clear from properties (i) and (iv) that this defines exactly one semantic rule for each attribute of production p that should have one. Moreover each other attribute occurs exactly once in a right-hand side of a semantic rule. For all

productions $Z \to w_0 X w_1$ with rule $q_0(Z(x_1)) \to aq(x_1)$, cf. (iii), the semantic rules are $s(Z) := sq(X)$ and $iq(X) := a$.

It is easy to see (or prove) from the construction of the attribute grammar G that for each complete derivation tree t of G, $D(t)$ consists of exactly one complete dependency path π. Moreover, if the homomorphism h is defined such that for each dependency edge e corresponding to a semantic rule $a(X_1) := v[b(X_2)]$, $h(e) = v$, then $h(\pi) = w$ where $q_0(t) \Rightarrow^* w$ by M. Finally, if $w = aw_1$, then the output tree into which t is translated is $w_1[a]$, and so the corresponding (unique) path through the output tree is (isomorphic to) $w_1^R a = w^R$, the reverse of w. From these considerations it follows that $M(L) = h(\text{cdpl}(G))$, and $M(L)$ is the reverse of $\pi(\text{OUT}(G))$. Hence $M(L) \in \text{HOM}(\text{CDPL}(\text{AG}))$, and $M(L)^R \in \pi(\text{OUT}(\text{AG, TREES}))$. Since $yT_{\text{fc}}(\text{RECOG})$ is closed under reverse, this proves the inclusions. ∎

5.8. EXAMPLE. Consider the yT-transducer M_k which, working on the derivation trees of the context-free grammar G_k, produces the output language $\{(a^n b)^{2k} \mid n \geqslant 1\}$, where $k \geqslant 1$. G_k has productions $Z \to A$, $A \to B$, $B \to B$, $B \to C$, and $C \to \lambda$. M has states $q_0, q_1, \ldots, q_k$ and the following rules (where x denotes x_1).

$$
\begin{aligned}
&q_0(Z(x)) \to aq_0(x),\\
&q_0(A(x)) \to q_1(x)\, bq_2(x)\, b \cdots q_k(x)\, b,\\
&q_j(B(x)) \to aq_j(x)\, a \qquad \text{for} \quad 1 \leqslant j \leqslant k,\\
&q_j(C(x)) \to aba \qquad \text{for} \quad 2 \leqslant j \leqslant k,\\
&q_1(C(x)) \to ba.
\end{aligned}
$$

Clearly M_k is k-copying and satisfies requirements (i)–(iv) in the proof of Theorem 5.7. In particular, cf. point (iv), with Z, A, B and C are associated the state-sequences $\langle q_0 \rangle$, $\langle q_0 \rangle$, $\langle q_1, \ldots, q_k \rangle$, and $\langle q_1, \ldots, q_k \rangle$, respectively. In the AG G_k, constructed as in the proof of Theorem 5.7, Z has the designated s-attribute s, A has i-attribute i_0 and s-attribute s_0, and B and C have i-attributes $i_1, \ldots, i_k$ and s-attributes $s_1, \ldots, s_k$. The semantic rules are indicated in the dependency graphs of Fig. 11, labeling the edge from c to d by v if the semantic rule is $c := v[d]$.

Note that G_k is (simple) k-pass: in the first pass $i_0(A)$, $i_1(B)$, $s_1(B)$, $i_1(C)$, and $s_1(C)$ can be computed, in the jth pass $(1 < j < k)$ $i_j(B)$, $s_j(B)$, $i_j(C)$, and $s_j(C)$ can be computed, and in the k-th pass $i_k(B)$, $s_k(B)$, $i_k(C)$, $s_k(C)$, $s_0(A)$, and $s(Z)$. ∎

As noted before, path languages can be used to compare the classes of output tree languages of different classes of AG, and thus compare their formal power. As an example, since $\pi(\text{out}(L\text{-AG, TREES}))$ is the class of context-free languages (Engelfriet and Filè, 1979), Theorem 5.7 shows that

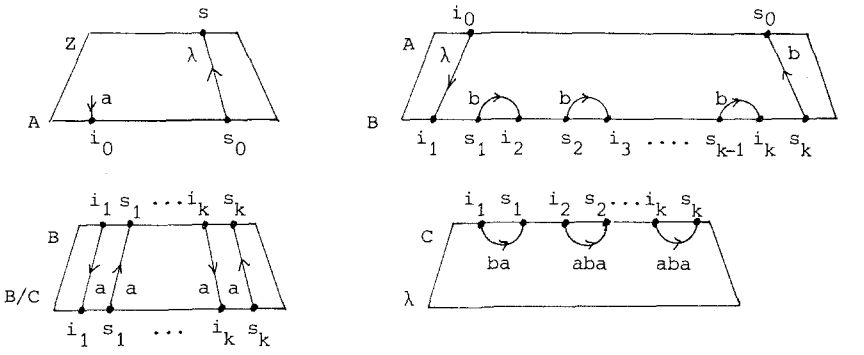

FIG. 11. Attribute grammar corresponding to yT-transducer.

OUT(L-AG, TREES)$\subsetneq$ OUT(AG, TREES) and so arbitrary AG are more powerful (schematically) than L-AG, cf. Lewis *et al.* (1974). In Engelfriet (1980) the "macro tree transducer" is introduced which is more powerful than the attribute grammar, because it can produce the path language $\{a^{2^n} \mid n \geqslant 0\}$ which is not in yT_{fc}(RECOG). We now show how the approach of this section can be used to prove that attribute grammars form a proper hierarchy with respect to the number of passes or the number of visits allowed.

An attribute grammar is (pure) *k-visit* if its attributes can be evaluated by a walk through the derivation tree (from root to root) in such a way that each subtree is visited at most k times (cf. Riis and Skyum (1980); for the notion of simple k-visit, see Engelfriet and Filè (1980)). It can be shown rather easily that every noncircular AG is k-visit for some k (see Riis and Skyum (1980)). Let us denote the class of pure k-visit AG by *PkV*-AG, and the class of pure k-pass AG by *PkP*-AG. Note that *PkP*-AG $\subseteq$ *PkV*-AG, and $\bigcup\{PkV\text{-AG} \mid k \geqslant 1\}$ is the class of all (noncircular) AG.

5.9. THEOREM. *For every* $k \geqslant 1$,

$$\pi(OUT(PkV\text{-}AG, TREES)) = yT_{fc(k)}(RECOG).$$

Proof. If G is k-visit, then the grammar G' constructed in the beginning of the proof of Theorem 5.6 is still k-visit (at the first visit the value of i_0 is computed). It should be clear that every *is*-path of a k-visit AG is itself k-visit (Definition 4.6); note that this does not hold vice versa! Hence, by Lemma 5.1, the yT-transducer M constructed from G' is k-copying. Consequently, by the proofs of Theorems 5.6 and 5.7, $\pi(\text{OUT}(G)) \in yT_{fc(k)}(\text{RECOG})$.

Vice versa, if in the second part of the proof of Theorem 5.7 M is k-copying, then the corresponding AG G is k-visit, because the unique complete path in the dependency network is k-visit.

This shows that $\pi(\text{OUT}(PkV\text{-AG, TREES})) = yT_{\text{fc}(k)}(\text{RECOG})$. ∎

Since it is proved in Engelfriet *et al.* (1980) that the classes $\{yT_{\text{fc}(k)}(\text{RECOG})\}_{k\geqslant 1}$ form a proper hierarchy, it follows directly from Theorem 5.9 that also the classes $\{\text{OUT}(PkV\text{-AG, TREES})\}_{k\geqslant 1}$ form a proper hierarchy. In particular (Engelfriet *et al.*, 1980) the language $\{(a^n b)^{2k} \mid n \geqslant 1\}$ is not in $yT_{\text{fc}(k-1)}(\text{RECOG})$; since, by Example 5.8, this language (or its reverse) is in $\pi(\text{OUT}(PkP\text{-AG, TREES}))$, this implies that also the classes $\{\text{OUT}(PkP\text{-AG, TREES})\}_{k\geqslant 1}$ form a proper hierarchy.

Finally we will show that two visits are more powerful than any number of passes, i.e., not every AG-translation can be realized by a multi-pass AG (independent of the semantics). To prove this we need a result similar to Theorem 5.7 for pure multi-pass AG; to characterize the output path languages of pure multi-pass AG it is more convenient to use the tree-walking automaton of Aho and Ullman (1971) and Engelfriet *et al.* (1980) than the top-down tree-to-string transducer. We need some definitions.

A *tree-walking transducer* (called *ct*-transducer in Engelfriet *et al.* (1980)) is a construct $M = (Q, \Sigma, \Delta, \delta, q_0, F)$, where Q is a finite set of states, Σ is the ranked input alphabet, Δ is the output alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a mapping from $Q \times \Sigma$ into the finite subsets of $Q \times D \times \Delta^*$, where $D = \{up\} \cup \{\text{down}(i) \mid i \geqslant 1\}$. Intuitively, a configuration of M consists of an input tree, an output tape, and a finite control with one pointer to a node of the input tree and another pointer to the end of the output tape. If the node x of the input tree pointed at is labeled $\sigma \in \Sigma$, M is in state q, and $\delta(q, \sigma)$ contains (q', d, w), then M can go into state q', add w to the output, and move to the father of x (if $d = up$) or move to the ith son of x (if $d = \text{down}(i)$). A succesful computation of M on a tree t should start at the root of t in state q_0 and should end at the root of t in a final state. Thus M realizes a tree-to-string translation. We denote by TWT the class of translations of tree-walking transducers. Note that $\text{TWT}(\text{RECOG}) \subseteq yT(\text{RECOG})$ and $yT_{\text{fc}}(\text{RECOG}) = \text{TWT}_{\text{fc}}(\text{RECOG})$, where fc indicates that the tree-walking transducer is "finite-crossing" or "finite-visit" (see Engelfriet *et al.* (1980)).

Now, corresponding to pure multi-pass AG, we define a restriction on tree-walking transducers which bounds the number of right-to-left moves on the input tree. An *R-move* consists of two consecutive moves: the first from the jth son of some node x up to x itself and the second down to the ith son of x, with $i \leqslant j$. Thus in an R-move the tree-walking transducer goes from a node to one of its left-brothers (or itself). A tree-walking transducer M is *R-bounded* with bound k if in every successful computation of M on an input

tree the number of R-moves is at most k. We denote by RTWT the class of translations realized by R-bounded tree-walking transducers. The inclusion in the next theorem is actually an equality.

5.10. THEOREM. *For every* $k \geqslant 1$, $\pi(\mathrm{OUT}(PkP\text{-AG}, \mathrm{TREES})) \subseteq \mathrm{RTWT}(\mathrm{RECOG})$.

Proof. First, it is obvious that $\mathrm{CDPL}(PkP\text{-AG}) \subseteq \mathrm{RTWT}(\mathrm{RECOG})$. In fact, it is easy to define a (nondeterministic) tree-walking transducer M which follows a complete dependency path in the dependency network of the input tree, and produces the path as output. Since the R-moves of M correspond precisely to the R-edges of the path, M is R-bounded with bound k. Second, it is easy to prove that RTWT(RECOG) is closed under reversal and *gsm*-mappings. Exactly as in the proof of Theorem 5.7, this implies the result. ∎

The next lemma provides a method to obtain languages outside $\pi(\mathrm{OUT}(PkP\text{-AG}, \mathrm{TREES}))$. The result and its proof are analogous to those concerning "meta-linear" top-down tree-to-string transducers in Theorem 3.2.10 and Corollary 3.2.12 of Engelfriet *et al.* (1980).

5.11. LEMMA. *Let L be a language over an alphabet Ω and let $\#$ be a symbol not in Ω. If $(L\#)^* \in RTWT(RECOG)$, then L is a context-free language.*

Proof. Let $M \in \mathrm{RTWT}$ and $L' \in \mathrm{RECOG}$ be such that $M(L') = (L\#)^*$. We first claim the following.

Claim. For every string $y \in L$ there is a successful computation of M on some tree t generating output of the form $u\#y\#v$ with $u, v \in (\Omega \cup \{\#\})^*$ such that during the generation of $\#y\#$ no R-moves are made by M.

In fact suppose that there is some $y_0 \in L$ for which this does not hold and consider any string of the form $(y_0\#)^n \in M(L')$. Then M can generate at most one $\#$ during each sequence of non-R-moves and hence $n \leqslant k+1$ where k is the bound on the R-moves of M. This is a contradiction and proves the claim.

We now construct a new tree-walking transducer M' which simulates all subcomputations of M which have no R-moves. To do this M' has to know, for every node x of a tree t, firstly, which are the states in which M may reach x starting from the root of t in its initial state, and secondly, in which states M may start from x and reach the root of t in a final state. Let us call these states the initial states and the final states at x, respectively. It is left as an exercise to the reader to show that there is an $L'' \in \mathrm{RECOG}$ which contains the trees of L' with this additional information labeled at each node. M' makes the following computations on a tree t. It first walks down to

some node x of t without generating output. Then it starts simulating M in one of the possible initial states at x. M' simulates M for some time, not using any of its R-moves, and ends the simulation at some node x' of t in a possible final state at x'. Finally, M' moves up to the root of t without generating output. It should be clear that $L = g(M'(L''))$, where g is the *gsm*-mapping $\{(w, y) \mid y \in \Omega^* \text{ and } w = w_1 \# y \# w_2 \text{ for some } w_1, w_2 \in (\Omega \cup \{\#\})^*\}$. Thus it remains to show that $M'(L'')$ is a context-free language. Since M' makes no R-moves, it visits each node at most once. Hence it can be simulated by a linear (non-copying) top-down tree-to-string transducer, cf. Theorem 4.9 of Engelfriet *et al.* (1980). Thus, since RECOG is closed under linear tree transducers, $M'(L'')$ is the yield of a recognizable tree language and hence context-free. ∎

It should be clear from these results that there is a very close relationship between evaluation strategies of attribute grammars and restricted types of tree-walking transducers.

We now state our main result on the power of passes and visits.

5.12. THEOREM. *The classes* $\{OUT(PkV\text{-}AG, TREES)\}_{k \geqslant 1}$ *form a proper hierarchy, and the same holds for* $\{OUT(PkP\text{-}AG, TREES)\}_{k \geqslant 1}$. *More precisely, the latter is a "small hierarchy" inside the first: for each* $k \geqslant 2$ *there is a tree language in* $OUT(PkP\text{-}AG, TREES)$ *which is not in* $OUT(P(k-1)\,V\text{-}AG, TREES)$, *and there is a tree language in* $OUT(P2V\text{-}AG, TREES)$ *which is not in* $OUT(PkP\text{-}AG, TREES)$ *for any* k.

Proof. It suffices to prove the statement of the theorem for the corresponding classes of path-languages. As noted before, the reverse of the language $\{(a^n b)^{2k} \mid n \geqslant 1\}$ is in $\pi(\text{OUT}(PkP\text{-AG, TREES}))$ by Example 5.8, but not in $yT_{fc(k-1)}(\text{RECOG})$ by Engelfriet *et al.* (1980), and hence not in $\pi(\text{OUT}(P(k-1)\,V\text{-AG, TREES}))$ by Theorem 5.9.

In Engelfriet *et al.* (1980) it is shown that the language $\{a^n b^n c^n \# \mid n \geqslant 0\}^*$ is in $yT_{fc(2)}(\text{RECOG})$, and hence (by Theorem 5.9) it is in $\pi(\text{OUT}(P2V\text{-AG, TREES}))$, see also the next example. However, by Lemma 5.11, this language is not in RTWT(RECOG), and hence, by Theorem 5.10, not in $\pi(\text{OUT}(PkP\text{-AG, TREES}))$ for any k. ∎

5.13. EXAMPLE. In this example we define a 2-visit AG G such that $h(cdpl(G)) = \{a^n b^n c^n \# \mid n \geqslant 0\}^*$ for some homomorphism h, cf. the proof of Theorem 5.10. This means that this AG is inherently non-multi-pass. In fact, if we turn G into a tree-valued AG in the obvious way, then its monadic output tree language $\{\#c^n b^n a^n \mid n \geqslant 0\}^*$ cannot be obtained by any multi-pass tree-valued AG.

G has productions $Z \to H$, $H \to H$, $H \to A$, $A \to A$, $A \to H$, and $A \to a$ with the dependency graphs of Fig. 12. Z has one s-attribute, A and H have two i-

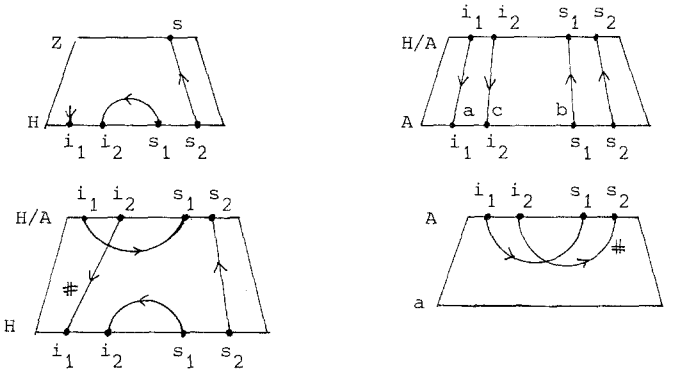

FIG. 12. An inherently non-multi-pass AG.

attributes and two s-attributes. The nonlabeled dependency edges are supposed to be labeled with λ; the labeling indicates the homomorphism h as in Fig. 11. An arbitrary monadic derivation tree $ZHA^{n_1}HA^{n_2}H \cdots HA^{n_k}a$ has a dependency network consisting of one complete dependency path π, such that $h(\pi) = a^{n_1}b^{n_1}c^{n_1}\#a^{n_2}b^{n_2}c^{n_2}\# \cdots a^{n_k}b^{n_k}c^{n_k}\#$; π visits each group A^n twice; at the first visit it generates a^n downwards and b^n upwards, and at the second visit it generates c^n downwards and continues with the next group of A's. ∎

Let us shortly consider the simple case again. It follows from Theorem 2.6 that the classes of tree translations of the pure and simple k-pass AG are equal, and hence OUT(PkP-AG, TREES) = OUT(SkP-AG, TREES), where the S stands for "simple." Also for visits there is no difference between simple and pure for the output tree language classes, cf. Engelfriet and Filè (1980). Hence Theorem 5.12 says that, independent of the simple/pure distinction, k passes are more powerful than $k-1$ visits, and two visits more powerful than any number of passes.

We note that similar results on passes and visits have been shown in (Riis, 1980; Riis and Skyum 1980) for the case of the tree translations realized by the different types of AG. It is not clear at all whether the path-approach can also be used to decide the k-visit property (as we did for passes in Section 4). In Riis and Skyum (1980) decidability of the k-visit property is proved by other means.

We finally note that by restricting the underlying context-free grammar of an AG to be linear, as in Examples 5.8 and 5.13, all results will stay valid, if one also restricts the yT-transducer to monadic input trees. Thus, the class of output path languages of these linear AG is equal to the class of output languages of 2-way deterministic *gsm* (or, ETOL_{FIN}, the class of finite index ETOL languages), see Engelfriet *et al.* (1980).

Conclusion

The formal power of attribute grammars lies in their dependency networks. In fact, first, dependency networks (even of L-AG) are able to simulate computations of exponential-time Turing machines, cf. Section 3. Second, the meaning of a derivation tree is a (tree) homomorphic image of the unraveled dependency network of the derivation tree (assuming that each node of the network is labeled with the appropriate semantic rule), cf. the proof of Theorem 5.7. Third, the possible evaluation strategies of an AG are determined by properties of its dependency networks; in particular, applicability of the pure multi-pass strategies is determined by the R-edges in the networks, cf. Sections 2 and 4. In this paper we have concentrated on dependency paths through networks and we have shown that path-properties of dependency networks are sufficient to deal with the formal power of pure multi-pass attribute grammars.

Acknowledgments

We are grateful to Henk Alblas for giving us the topic of the paper and for many stimulating discussions. We thank Erik Meineche Schmidt and Sven Skyum for pointing out the relevance of Chomsky Normal Form for attribute grammars.

Received: February 12, 1981; revised: July 23, 1981

References

Aho, A. V. and Ullman, J. D. (1971), Translations on a context-free grammar, *Inform. Contr.* **19**, 439–475.

Alblas, H. (1980), "A Characterization of Attribute Evaluation in Passes," Memorandum 315, Twente University of Technology; *Acta Inform.*, in press.

Bochmann, G. V. (1976), Semantic evaluation from left to right; *Comm. ACM* **19**, 55–62.

Chandra, A. K., Kozen, D. C., and Stockmeyer, L. J. (1981), Alternation, *J. Assoc. Comput. Mach.* **28**, 114–133.

Cook, S. A. (1971), Characterizations of pushdown machines in terms of time-bounded computers; *J. Assoc. Comput. Mach.* **18**, 4–18.

Engelfriet, J. (1975), Bottom-up and top-down tree transformations—a comparison; *Math. Systems Theory* **9**, 198–231.

Engelfriet, J. (1980), Some open questions and recent results on tree transducers and tree languages, *in* "Formal Language Theory; Perspectives and Open Problems" (R. V. Book, Ed.), pp. 241–286, Academic Press, New York.

Engelfriet, J. and Filè, G. (1979), "The Formal Power of One-Visit Attribute Grammars," Memorandum 286, Twente University of Technology; *Acta Inform.*, in press.

Engelfriet, J. and Filè, G. (1980), "Simple Multi-Visit Attribute Grammars," Memorandum 314, Twente University of Technology.

ENGELFRIET, J., ROZENBERG, G., AND SLUTZKI, G. (1980), Tree transducers, L systems and two-way machines; *J. Comput. System Sci.* **20**, 150–202.

ENGELFRIET, J. AND SLUTZKI, G. (1979), Bounded nesting in macro grammars, *Inform. Contr.* **42**, 157–193.

JAZAYERI, M., OGDEN, W. F., AND ROUNDS, W. C. (1975), The intrinsically exponential complexity of the circularity problem for attribute grammars, *Comm. ACM* **18**, 697–706.

JONES, N. D. (1980), "Circularity Testing of Attribute Grammars Requires Exponential Time: A Simpler Proof," DAIMI PB-107, Aarhus University.

KASTENS, U. (1980), Ordered attributed grammars, *Acta Inform.* **13**, 229–256.

KENNEDY, K. AND WARREN, S. K. (1976), Automatic generation of efficient evaluators for attribute grammars, *in* "Conf. Record of 3d Symp. on Principles of Programming Languages," pp. 32–49.

KNUTH, D. E. (1968), Semantics of context-free languages, *Math. Systems Theory* **2**, 127–145. Correction: *Math. Systems Theory* **5**(1971), 95–96.

LADNER, R. E., LIPTON, R. J., AND STOCKMEYER, L. J. (1978), Alternating pushdown automata, *in* "Proceedings, 19th Ann. Symp. on Foundations of Computer Science," pp. 92–106.

LEWIS, P. M., ROSENKRANTZ, P. J., AND STEARNS, R. E. (1974), Attributed translations, *J. Comput. System Sci.* **9**, 279–307.

RÄIHÄ, K.-J. AND UKKONEN, E. (1980), On the optimal assignment of attributes to passes in multi-pass attribute evaluators, *in* "Proceedings, 7th ICALP, Noordwijkerhout," Lecture Notes in Computer Science No. 85, pp. 500–511, Springer-Verlag, Berlin.

RIIS, H. (1980), "Subclasses of Attribute Grammars," DAIMI PB-114, Aarhus University.

RIIS, H. AND SKYUM, S. (1980), "k-Visit Attribute Grammars," DAIMI PB-121, Aarhus University; *Math. Systems Theory*, in press.

ROUNDS, W. C. (1970a), Mappings and grammars on trees, *Math. Systems Theory* **4**, 257–287.

ROUNDS, W. C. (1970b), Tree-oriented proofs of some theorems on context-free and indexed languages, *in* "Proceedings, 2nd Ann. ACM Symp. on Theory of Computing, Northampton, Mass.," pp. 109–116.

RUZZO, W. L. (1980), Tree-size bounded alternation, *J. Comput. System Sci.* **21**, 218–235.

THATCHER, J. W. (1973), Tree automata: An informal survey, *in* "Currents in the Theory of Computing" (A. V. Aho, Ed.), Prentice Hall, Engelwood Cliffs, N.J.

JAZAYERI, M. AND WALTER, K. G. (1975), Alternating semantic evaluator, *in* "Proceedings, ACM 1975 Ann. Conf.," pp. 230–234.

Printed in Belgium